apple® logo primer

gary g. bitter
nancy ralph watson

Frequently Used Turtle Commands

ARCLEFT (ARCL)	Turtle draws arc to left according to inputs, p. 23
ARCL1	Turtle draws arcs or circles to left according to inputs, p. 25
ARCRIGHT (ARCR)	Turtle draws arc to right according to inputs, p. 23
ARCR1	Turtle draws arcs or circles to right according to inputs, p. 25
CIRCLEL	Turtle draws circle to left according to input, p. 24
CIRCLER	Turtle draws circle to right according to input, p. 24
BACK (BK)	Moves turtle backwards, p. 8
DOT	Places dot on screen in location indicated by *x* and *y* coordinates inside brackets, p. 54
END	Indicates completion of procedure in edit mode, p. 18
FENCE	Restricts turtle from moving beyond screen boundaries, p. 52
FORWARD (FD)	Moves turtle forward, p. 8
HIDETURTLE (HT)	Turtle disappears, p. 13
LEFT (LT)	Turns turtle to left, p. 8
PENDOWN (PD)	Pen position in which turtle draws, p. 11
PENERASE (PE)	Pen position that erases lines that turtle moves over, p. 12
PENREVERSE (PX)	Pen does opposite of what turtle moves over, p. 12
PENUP (PU)	Lifts pen so that no drawing occurs, p. 11
REPEAT	Instructions are to be executed more than once, p. 11
RIGHT (RT)	Turns turtle to right, p. 8
SETBG	Sets screen color according to code selected, p. 14
SETHEADING (SETH)	Turtle faces specific direction indicated in degrees, p. 57
SETH TOWARDS	Turtle faces direction indicated by *x* and *y* coordinates, p. 58
SETPC	Sets pen color according to code selected, p. 14
SETPOS	Moves turtle to location indicated by *x* and *y* coordinates, p. 55
SETX	Moves turtle to *x*-coordinate location indicated by input, p. 56
SETY	Moves turtle to *y*-coordinate location indicated by input, p.56
SHOWTURTLE (ST)	Turtle appears on screen, p. 7
WINDOW	Allows turtle to go off screen without wrapping, p. 52
WRAP	Turtle keeps coming back on screen when its moves exceed screen limits, p. 51

Frequently Used File Commands

CATALOG	Lists all files on disk, p. 34
EDIT (ED)	Places named procedure in edit mode, p. 41
ERASE (ER)	Deletes procedures in memory, p. 37
ERASEALL (ERALL)	Deletes procedures in memory, p. 38
ERASEFILE	Deletes files from disk, p. 45
LOAD	Places named file from disk into memory, p. 38
PACKAGE	Places named procedures in their own file, p. 36
PRINTOUT (PO)	Displays named procedure on screen, p. 22
PRINTOUT PROCEDURES (POPS)	All procedures in memory are displayed on screen, p. 22
PRINTOUT TITLES (POTS)	Titles of procedures in memory are displayed on screen, p. 22
SAVE	Saves named procedure on disk, p. 35

Apple® Logo Primer

Apple® Logo Primer

Gary G. Bitter
Nancy Ralph Watson
Arizona State University

Reston Publishing Company, Inc.
A Prentice-Hall Company
Reston, Virginia

Library of Congress Cataloging in Publication Data
Bitter, Gary G.
Apple Logo primer.

Includes index.
1. Apple computer—Programming. 2. LOGO (Computer program language) I. Watson, Nancy. II. Title.
QA76.8.A66B56 1983 001.64'2 83-8699
ISBN 0-8359-0314-1 (pbk.)

Photographs by Ralph Ferguson

Apple® is a registered trademark of Apple Computer, Inc.

© 1983 by Reston Publishing Company, Inc.
A Prentice-Hall Company
Reston, Virginia

All rights reserved. No part of this book may be reproduced in any way, or by any means, without permission in writing from the publisher.

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

To everyone,
young and old,
who loves turtles

Contents

Preface ..... **xi**

Acknowledgments ..... **xiii**

I Step-by-Step Logo **1**

1 What's Needed to Begin **2**
The Bare Necessities, 2

2 Starting with the Turtle **3**
Loading Logo into Your Computer, 3 / Exploring with the Turtle, 3 / Correcting Typing Errors, 5 / Screen Characteristics, 7 / CLEARSCREEN or CS, 7 / SHOWTURTLE or ST, 7 / Turtle Graphics, 7 / Turtle Tricks, 9 / Manipulating the Pen, 11 / PENUP or PU, 11 / PENDOWN or PD, 12 / PENREVERSE or PX, 12 / PENERASE or PE, 12 / Manipulating the Turtle, 13 / HIDETURTLE or HT, 13 / Color at Last, 13 / Black and White Monitors, 15

3 Developing a Procedure **16**
More Screen Modes, 16 / SPLITSCREEN or CTRL S, 16 / FULLSCREEN or CTRL L, 16 / TEXTSCREEN or CTRL T, 16 / CLEARTEXT, 17 / HOME, 17, Procedures, 17 / TO, 18 / Instructions, 18 / END, 18 / More on Procedures, 19 / Procedures within Procedures, 20 / PRINTOUT Procedures (POPS), 22 / POTS, 22 / PO, 22 / Stored Procedures, 23

4 REPEAT **27**
Saving Time with REPEAT, 27 / REPEAT, 27 / Repeating Procedures, 30 / Balloon, 31 / Balloon Bouquet, 32

5 Loading and Saving Programs **34**
Saving Procedures, 35 / Saving Procedures in Packages, 36 / ERASE and ERASEALL, 37 / Is It Really Saved?, 38 / CATALOG, 38 / LOAD, 38 / Loading Procedures That Have Been Saved, 39 / POPS, POTS, PO, 39

6 **Editing** 41
The Edit Mode, 41 / Control Characters, 41 / Using the Editor, 43 / Saving Your Edited Procedure, 44 / Editing Saved Procedures, 44 / Editing Procedures Saved More Than Once, 45 / Deleting Procedures from the Disk, 46

7 **Recursive and Coordinates** 48
A Little Magic, 48 / Recursive, 49 / CTRL G, 49 / CTRL W, 49 / CTRL Z, 49 / Setting Limits for the Screen, 51 / FENCE Command, 52 / WINDOW Command, 52 / Now We're Getting Somewhere, 53 / DOT Command, 54 / SETPOS Command, 55 / More with X and Y, 56 / SETHEADING Command, 57 / SETHEADING TOWARDS Command, 58

8 **Combining Procedures** 59
Combining Procedures, 59 / CHAINLINK, 59 / Refining Your Skills, 61 / Activity 1—ROOM, 62 / Activity 2—TABLE, 63 / Activity 3—PANE, 63 / Activities Using ROOM, 64 / Activity 4—SEASCAPE, 64 / Activity 5—COUNTRYROAD, 65

9 **Input and Variables** 66
Input Variables, 66 / Multiple Inputs, 68 / Procedures within Procedures Using Inputs, 70 / Internally Generated Input, 71 / POS, 72 / XCOR and YCOR, 73 / Making Use of Internally Generated Input, 74 / Using MAKE in a Procedure, 76

10 **Arithmetic Operations** 79
Logo and Arithmetic, 79 / Using Arithmetic Operations in Procedures, 80 / Random as an Arithmetic Operation, 83 / Using Random Numbers in Procedures, 84 / Other Arithmetic Operations, 86

11 **Conditionals, IF, AND, and OR** 89
Conditionals, 89 / A Different Use of Conditionals, 90 / IF with OR, 92 / IF and AND, 94

12 Animation **95**
Simulating Movement, 95 / The Moving Truck, 95 / An Underwater Scene, 96 / The Hungry Worm, 98

13 Text Use **100**
Some Words about Words, 100 / PRINT with Text, 101 / "WORD, 101 / [WORD], 102 / :WORD, 102 / Another Use for Brackets, 102 / Manipulating the Text, 103 / Counting the Elements, 104 / WORD and SENTENCE Operations, 104 / READCHAR and READLIST Operations, 104

14 Combining Text and Graphics **106**
Programs Using Text and Graphics, 106 / The Game, 108 / First Question, 108 / Second Question, 108 / Third Question, 108 / Discussion, 109

II Quick-Start Logo **111**
Getting Started, 112 / Turtle Steps, 112 / Repeating Yourself, 114 / Fooling Around with the Turtle, 115 / PENERASE, 116 / PENREVERSE, 116 / Keys That Control the Cursor, 117 / Keys and Words That Control the Screen, 118 / CTRL L (FULLSCREEN), 118 / CTRL S (SPLITSCREEN), 118 / CTRL T (TEXTSCREEN), 118 / HOME, 119 / Keys That Control the Turtle, 119 / CTRL G, 119 / CTRL W, 120 / Adding a Splash of Color, 120 / The Black and White Monitor, 121 / Placing Programs into Memory, 122 / Procedures within Procedures, 124 / Editing Your Procedures, 125 / Initializing a Disk, 126 / Preparing or Initializing a Disk to Save Your Programs, 126 / How To Save Your Programs, 127 / Saving, 127 / Saving One Program, 127 / Saving Several Procedures, 128 / Erasing Procedures, 129 / PACKAGE Command, 129 / POTS, POPS, PO, 131 / A Trip to the Library, 131

III An Overview of Logo **135**
Definition, Origins, History, 137 / A Definition of Logo, 137 / The Origins of Logo, 137 / History of Logo, 139 / The Language of Logo, 140 / Definition of Turtles, 140 / The Functions of Turtles, 141 / Children and Logo, Case Study, 142 / Children as Teachers, 144 / Children as Epistemologists, 145 / Logo as an Educational Tool, 146 / Turtle Geometry, 147 / Logo in Art, 147 / Logo and Social Skills, 147 / Logo in Special Education, 149 / Logo in Adult Education, 150 / Logo in the Home—Children and Adults, 151 / More About Logo, 152 / Other Modes of Logo, 152 / The Nuts and Bolts of Logo, 152 / Seven Major Versions of Logo, 153 / The School of the Future, 154 / Summary, 155

Appendixes **157**
I Answers to Practice Activities **157**
II Suggested Activities **197**

Glossary/Index **199**

Preface

*Apple® Logo Primer** has been written to make Logo easy to understand for anyone. We prepared a careful instructional sequence for the novice. But we didn't forget the advanced user: We also included a section to introduce those with previous Logo knowledge to Apple Logo quickly.

The book is divided into three distinct parts. Part I is a step-by-step development of Logo with no user prerequisites except access to an Apple computer with Apple Logo. Part II, called Quick-Start Logo, is for the person who has some familiarity with Logo. Part III includes a broad overview of Logo, including the history and applications.

For convenience we also included an outline of commands and error messages (inside the covers) and a detailed glossary/index. Our intent is to make this book a guarantee for success with Logo, as well as a resource book with features to make answers to questions easier to find and to understand.

Here are the contents of each part of the book. Part I is a carefully developed presentation of the concepts of Apple Logo which we feel are the most useful for education. Since the length of this book was limited, some topics had to be covered only briefly or eliminated. Refer to the manuals which come with Apple Logo for a listing of all of the capabilities.

Part I has fourteen chapters. The topics in Chapters 1 and 2 include getting started, using the turtle, screen commands, and developing programs. Chapter 3 includes additional screen commands, procedures, and the Logo library; Chapter 4 outlines the use of the REPEAT capabilities. Chapter 5 covers all the information about using a disk to load and save your masterpieces. Chapter 6 deals with the editing and special save functions. Recursion and the use of coordinates and heading are discussed in Chapter 7; Chapter 8 continues with procedures within procedures. Input and variables are included in Chapter 9; and arithmetic operations and random numbers are introduced in Chapter 10. Conditionals, IF, AND, and OR are covered in Chapter 11. Animation—yes, you do your own—is presented in Chapter

*Apple® is a registered trademark of Apple Computer, Inc.

12. Chapter 13 is the introduction to text use; and Chapter 14 concludes with a program combining text and graphics.

Part II provides a person with previous programming or Logo experience an opportunity to learn Apple Logo without reading through all the details. This section includes explanations of the commands and operations of Logo, plus examples of the features and practice activities. The topics covered are similar to those in Part I, although not dealt with in as much detail.

Part III includes definitions, history, instructional uses, and the origins of Logo. Children and Logo are discussed, using a case study for illustration. Also outlined in this section is the use of Logo in the classroom, with turtle geometry and art used as examples. Included is a discussion of education-related applications to social skills, special education, adult education, and the home. Part III concludes with ideas about education of the future and a summary.

As you can see, the book has been carefully developed for the user. In other words, we tried to make this book "user friendly" with the inclusion of a glossary/index, commands and error messages (on the inside covers), and options for learning the language. *Apple® Logo Primer* attempts to make you a Logo user with a minimum of anxiety and frustration. We hope you find Logo an exciting, useful, and creative language.

Acknowledgments

Special thanks are due to those who supported the authors throughout the preparation of this manual:

- To Eric Olson, who freely shared his creativity and knowledge with us and was ready and willing whenever we needed him
- To Matthew Watson, whose persistence and experimentation flavors the graphics with youthful interpretations
- To Ralph Ferguson, whose patience, quest for a challenge, and sense of perfection lend invaluable visual representations to the learners of Logo
- To Meryl Thomas for her unending patience, understanding, and cheerful wake-up calls
- To Carol King, whose sincere interest in and encouragement of this project made it all happen.

Apple® Logo Primer

STEP-BY-STEP LOGO

Logo as an intricate, logical, creative language is defined in this section on a level geared toward those with little or no programming experience. It is also intended for those who desire to learn the secrets and idiosyncrasies of Logo which one must usually discover by chance, trial and error, or from someone already familiar with the language. By working through the frequent practice activities in each chapter, the opportunity to experience applications using Logo concepts is provided. The inside covers, appendixes, and glossary/index will be useful as resources as you learn Logo.

Throughout the text are informal questions of a predictive nature, the answers to which are often revealed in the sentences that follow. To make the best use of this discovery process, it is suggested that you discipline yourself to think through the questions before reading on. This serves as another self-check on your comprehension of the material.

1 What's Needed to Begin

THE BARE NECESSITIES

Before attempting to use Logo on your Apple®, it is wise to determine if your computer is equipped with everything it needs to handle the Logo software. Your Apple should have the following four items (in addition to the keyboard and monitor):

1. Disk drive
2. Logo diskette
3. Blank diskette (if you wish to save any of your Logo programs)
4. 64 K memory

If you do not have 64 K, you will be unable to use Logo. You can purchase extra memory at most stores which carry Apple computers.

Apple® is a registered trademark of Apple Computer, Inc.

Starting with the Turtle

2

Loading Logo into Your Computer

When you are ready to begin using Logo, simply follow these steps:

1. Insert the Logo disk into the disk drive.
2. Turn on the computer. The power switch is on the back left side of the machine.
3. Turn on the monitor (some call it the television).
4. When the light on the disk drive goes off, press the RETURN key. Don't worry about the messages on the screen regarding inserting your own disk. Press RETURN and Logo will be loaded into the computer.
5. The screen will now say

   ```
   WELCOME TO LOGO
   ?
   ```

   The cursor (flashing white box) will be on the right of the question mark.
6. Type SHOWTURTLE (or the abbreviation ST) and press RETURN. You are now ready to use Logo. As soon as you type SHOWTURTLE you will see the infamous turtle in the center of the screen.

Exploring with the Turtle

Logo works on the principle of giving a turtle (actually a small triangle-shaped object in the center of the screen) a pen to draw with. You will learn the turtle's vocabulary so that you can give instructions to it. Each tiny movement made by the turtle is called a *step*. The turtle can take steps either forward or backward. For example,

```
FORWARD 100     (The turtle moves forward 100 steps)
```

Forward can be in different directions depending on the direction the turtle is facing when you give the FORWARD command. Sometimes it is difficult to determine in which direction the turtle is heading. The point is all white—the base has a black area. When the turtle is at its "home" position, it is always facing or heading straight up, in the center of the screen.

```
BACK 150      (The turtle moves backward 150 steps)
```

After entering this line, the turtle is still facing straight up and does not change its direction to go backward, just like you would not turn around if told to walk backward.

Remember to always press the RETURN key each time you finish typing a line. Watch the turtle's movements carefully each time you press RETURN. If you make an error, simply retype the line.

In order to make the turtle change the direction in which it is facing or heading, as it is technically referred to, the two instructions are LEFT and RIGHT. Since the turtle begins by facing straight up, a command of

```
RIGHT 90
```

causes the turtle to change its heading by turning 90 degrees to the right. Now the turtle is heading towards the right of the screen, exactly a 90 degree turn.

```
LEFT 90
```

puts the turtle back at its original heading. By entering

```
LEFT 90
```

the turtle will now face the left side of the screen. Watch what happens when you enter this line:

```
RIGHT 180
```

Since there are 180 degrees in a half-turn, the turtle should now be heading to the right again. To get it back to the original position again, all that is needed is

```
LEFT 90
```

Combining the four instructions just introduced, a short program can be written. Enter these lines:

```
FORWARD 25    (Moves the turtle 25 steps forward)
RIGHT 90      (Turtle makes a 90° turn)
FORWARD 25    (Turtle goes another 25 steps forward)
LEFT 90       (Returns to the original heading)
BACK 25       (Facing up, but moving backward 25
               steps)
```

This is an example of a simple set of instructions. Did you notice how the turtle actually made the 90 degree turns when you pressed RETURN?

Enter these lines:

```
CLEARSCREEN   (Erases whatever graphics are on the
               screen)
RIGHT 45      (Turtle turns 45° to the right)
```

FORWARD 50	(Goes 50 steps forward)
LEFT 90	(Makes a 90° turn)
FORWARD 50	(50 more steps forward)
LEFT 90	(Turns left 90°)
BACK 50	(Steps backward 50 times)

Watching the turtle make each move assists you in visualizing just how the turtle operates.

Correcting Typing Errors

Each time you enter a line into the computer, it must be in the exact form specified, including correct spelling, spacing, and the use of special characters. If you notice that you have made a typing error before pressing RETURN, you can simply use the left arrow (near the right end of the second row of keys) to move the cursor to the left until you reach the position which you wish to change. Simply type the corrected line and press RETURN. For example,

RIGT 90	(Don't press RETURN)

While your cursor is still on this line, you have discovered the misspelling of RIGHT. Look on the right side of the keyboard and find the two arrows. Press the arrow pointing to the left, moving the cursor back across what you have typed, erasing it, until you erase the T. At this time you need only to type the correction (HT) and complete the line by typing a space followed by 90. Give it a try.

CTRL A (holding down the key marked CTRL and pressing the A) sends the cursor to the beginning of the line without erasing any of it. Enter the incorrect line again and a CTRL A. Proceed from there using the right arrow until you reach the place where the error occurred. Place the cursor over the character you want to replace, type the correction, then press CTRL D. If you want to add a left out character, place the cursor on the character or space following the spot where you wish to make an insertion and type the character.

Enter these lines and go back to correct them before pressing RETURN.

Error Line	*Correct Line*
LEF 45	LEFT 55
BAK 10P	BACK 10
FRWARD 90	FORWARD 190

If you do enter a line incorrectly and press RETURN before you discover your error, you will receive a tasteful

error message notifying you that the computer does not know how to do what you have asked. For example:

```
I DON'T KNOW HOW TO LEF
```

You will also receive a message if you omit the space between the command and the number. At this point you should check your last line to determine the error. The CTRL Y control character will reprint the last line you typed and allow you to edit it using the arrows and CTRL A. The cursor will be at the end of the line so CTRL A must be used to place the cursor at the beginning of the line. After making corrections, press RETURN. (An error is called a *bug* in computer lingo.) For example, if you typed

```
FORWRD 100
```

you would get this message:

```
I DON'T KNOW HOW TO FORWRD
```

This is called an *error message*. It means that the word you have entered is not in the turtle's vocabulary. To continue working with the turtle, just type the line over again. Remember that if you discovered your error *before* pressing RETURN, use the left arrow or CTRL A to move the cursor back to the spot where you made the mistake; type in the corrected line from that point and press RETURN. You will save yourself time by proofing each line before pressing RETURN.

Practice Activities

1. Try making the turtle go one step to get an idea of how far it is. Enter CS (the abbreviation of CLEARSCREEN) first.
2. When you instruct the turtle to make a turn, from what position does it begin?
3. In which direction is the turtle pointing when you give the command BACK?
4. Make the turtle turn 360 degrees without drawing anything.
5. Make the turtle go to the top of the screen, turn right, and end up at the top right corner.
6. How many turtle steps wide is the screen?
7. How many turtle steps high is the screen?

8. How many turtle steps make approximately 3 inches?
9. How many degrees would you turn the turtle to make it go completely around once without drawing anything?

Screen Characteristics

There are several different screen modes which are available to you, depending on what you are doing with the turtle. You must choose from one of them and type it into the computer. **Be sure to press RETURN after you complete typing each command to signal the computer you are finished.**

CLEARSCREEN or CS

The CLEARSCREEN command (CS for short) is self-explanatory: The screen is cleared of all graphics and the turtle is placed in its home position. Three lines of instructions, a question mark, and the cursor appear near the bottom. This command can also be used any time you wish to eliminate text or graphics on the screen. This was used in the short programs written earlier.

SHOWTURTLE or ST

By typing the SHOWTURTLE command (ST for short), the turtle will appear in the middle of the screen (coordinates 0,0). This is considered the home position for the turtle. The question mark and cursor will be located near the bottom left of the screen. When you first load Logo into your computer, no turtle will be visible. By typing SHOWTURTLE, there it is. This is where the turtle will always begin.

Turtle Graphics

Assuming you have loaded Logo, type one of the previous screen commands, SHOWTURTLE or CLEARSCREEN. The turtle is now at your disposal.

In the table that follows are the four different *primitives*, the name given to Logo's commands or basic instructions. You have been introduced to four of them—FORWARD, BACK, RIGHT, and LEFT. After studying each example, enter it into the computer to watch the turtle go to work. Don't forget to use the CLEARSCREEN (or CS) command when you want to start your design with a blank screen. Notice in the examples that next to each primitive there is an abbreviated form—you may use either the long or abbreviated form when giving instructions to the turtle.

FORWARD or FD	BACK or BK
The FORWARD command, followed by a space and a number, indicates the distance you wish the turtle to travel forward across the screen. Whatever direction the turtle is pointed is considered forward.	The BACK command, followed by a space and a number, indicates the distance you wish the turtle to move back.
Example:	*Example:*
FORWARD 40 or FD 40	BACK 60 or BK 60
RIGHT or RT	**LEFT or LT**
This command, followed by a space and a number, turns the turtle the number of degrees indicated to the right.	The left command, followed by a space and a number, turns the turtle to the left the number indicated.
Example:	*Example:*
RIGHT 90 or RT 90	LEFT 45 or LT 45

As you enter these instructions observe the turtle carefully and it will become apparent just how precisely the turtle is executing the commands.

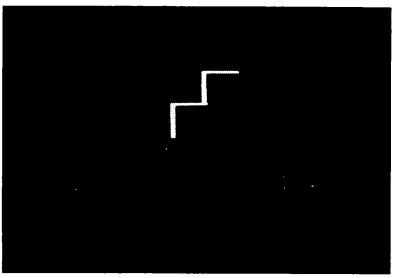
Practice Activities

1. Make a small square using the FORWARD and RIGHT commands.
2. Make the same square as in number 1, using the BACK and LEFT commands.
3. Design a rectangle that is very tall and skinny.
4. Design a rectangle that is short and wide.

Program 1: Begin by creating two stairs and use the abbreviations presented previously:

```
CS        (Starts you with a fresh drawing pad)
FD 25
RT 90
FD 25
LT 90
```

```
FD 25
RT 90
FD 25
```



What would your next instruction be in order to add more stairs? Add several more stairs until the turtle reaches the top of the screen.

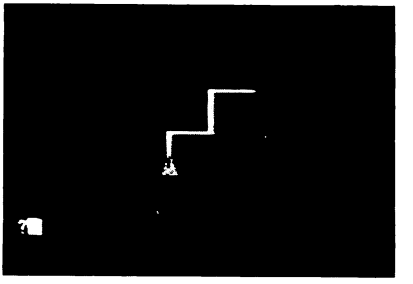
Practice Activities

1. Make a staircase that goes from the center of the screen down to the right.
2. Make a staircase going from the center down to the left.
3. Make a staircase with large steps.
4. Make a staircase with very small steps.

Turtle Tricks

Clear the screen and enter these lines:

```
RT 90
BK 25
LT 90
BK 25
RT 90
BK 25
LT 90
BK 25
```

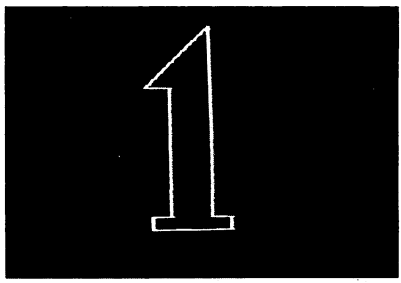


What is happening here? The stairs are drawn with the turtle facing backwards. Do you see the difference when RT and LT are switched in the two programs? It gets tricky when the turtle is headed backwards. You must always be aware of which direction it is facing.

Do the four exercises in the Practice Activities above using the BACK command instead of FORWARD.

Program 2: It's time to get more complicated. Enter the program shown below. It is a series of instructions which will make a familiar object. Attempt to predict what each instruction will do before the turtle moves. This is helpful in anticipating movements when you begin to design your own projects.

```
CS
BK 40
FD 100
LT 90
FD 20
RT 135
FD 75
RT 135
FD 154
LT 90
FD 15
RT 90
FD 10
RT 90
FD 63
RT 90
FD 10
RT 90
FD 16
```

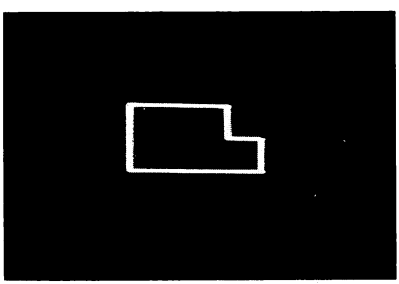


The graphic you entered is the number 1. You may notice some distortion on your screen causing the corners of the lines to appear as if they were going beyond the bounds of the object. This occurs on some monitors and not on others. Often by using colors this distortion will be eliminated. For now, don't worry about it as it does not indicate that your program is inaccurate.

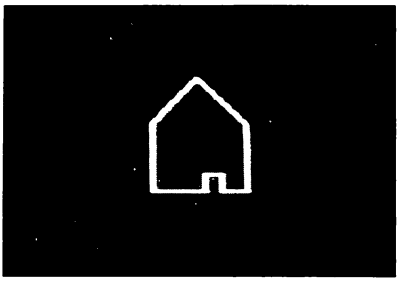
Practice Activities

Design a program to produce an object that looks like a

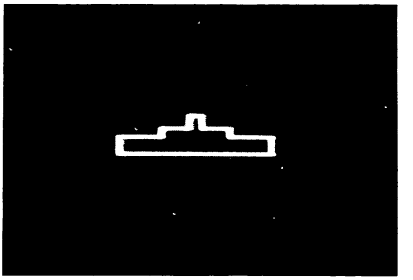
1. Truck



2. House



3. Submarine



Manipulating the Pen

There are several more commands which you will find are very useful when manipulating the turtle. After typing one, the turtle will assume this position until given a new command.

PENUP or PU

A self-explanatory term, PENUP simply allows you to manipulate the turtle without leaving a trail. In other words, it is as if you raised the pen from the paper. If you wish to move the turtle somewhere without drawing, use this command. After you have located the turtle in the spot where you intend to resume drawing, you may enter the next command. (Only by typing PENDOWN or PD—explained in the next paragraph—will the turtle begin drawing again.)

For example, enter these lines and observe the turtle's movements:

```
CS
PU      (Lifts the pen up so no drawing occurs)
FD 50   (Turtle moves without leaving a trail)
LT 45   (Heading changes)
BK 100  (Movement is backwards without a trail)
RT 120  (Turtle changes angle it is facing)
FD 50   (Movement is 50 steps forward)
```

Leave the turtle in this position.

PENDOWN or PD

PENDOWN (or PD) is the position the turtle is automatically in when Logo is loaded. The PD command must be used after the command PENUP when you are ready to begin drawing again. Check to determine if the turtle is heading in the direction you desire before giving it your next instruction. For example,

```
PD        (Places the pen back on the screen)
BK 50     (Turtle moves backward and drawing is occurring)
LT 120    (Heading is now 120°)
FD 100    (Movement is 100 steps forward)
RT 45     (Turtle makes a 45° turn)
BK 50     (Turtle moves back 50 steps)
```

The turtle just retraced its steps from the trail it followed with the pen up.

PENREVERSE or PX

The PENREVERSE command instructs the turtle to do exactly the opposite of whatever it is moving over. For example, if it moves over a line that has been drawn, it will erase it. If it goes over blank screen space, it will draw. If it crosses a line at an angle, it will erase only the small portion which it moves over.

```
CS
FD 100
LT 90
FD 60
LT 135
PX        (PENREVERSE)
FD 175    (Try to distinguish the tiny spot it erased when
           crossing the vertical line; it is difficult to see)
HOME
FD 50
```

PENERASE or PE

PENERASE (or PE) acts as an eraser because whatever you send the turtle over after entering the PE command will be "undrawn." If the turtle happens to cross a line at an angle, it will erase only one turtle step of that line, the one where it crosses the line. Enter PD before continuing.

Practice Activities

1. Make a square in the upper right-hand corner of the screen, without leaving a trail.
2. Draw half a square on the left-hand side of the screen and its mirror image on the right side of the screen.
3. Draw a rectangle and then erase it (don't use CS).

Manipulating the Turtle

HIDETURTLE or HT

Using the HIDETURTLE command causes the turtle to disappear from the screen. You may want to get rid of the turtle to get a better look at your design, or you may use this command at the beginning or end of your program so that your design is not distorted with the turtle in the middle of it.

For example, type in HIDETURTLE or HT and watch the screen. Without a point of reference, manipulating the turtle is more difficult and can be a real challenge in visualization. To see the turtle again, SHOWTURTLE must be entered.

SHOWTURTLE (or ST), which has already been introduced, is the opposite of HIDETURTLE. After the turtle is hidden, this command is necessary in order to have the turtle displayed on the screen.

Try some of the practice exercises that follow to become more familiar with the turtle. (You can not save your programs until you read the next section.) Remember to use the CLEARSCREEN (or CS) command to begin with a clean screen.

Practice Activities

1. Type ST and HT several times.
2. Type HT and attempt to draw a triangle.
3. Type HT and PU. Enter several commands to move the turtle around the screen. Place your finger on the spot where you think the turtle is located and then type in ST. How close were you?
4. Using a felt-tipped pen, mark a spot on the screen. Using PU and HT, place the turtle as close to the spot as possible.
5. Design a house with a door and a pointed roof. Then add windows and a chimney.
6. Attempt to create a two-dimensional object such as a box, room, or table.

Color at Last

Are you getting tired of black and white? Not to worry! (If you don't have a color monitor, refer to the next section.) Even though Apple Logo has a limited number of colors, they are delightful to play with after you've been using black and white for a while. The color code is as follows:

0 = Black	3 = Violet
1 = White	4 = Orange
2 = Green	5 = Blue

Two main commands control the color; one is for the pen and one is for the background.

SETPC 4	(Makes the pen write in color number 4—orange)
SETBG 1	(Makes the screen color number 1—white)

The commands must be placed carefully in the program and will remain until new colors are given. Be sure to indicate pen and background colors before giving drawing instructions.

```
CS
SETPC 1
SETBG 5
BK 50
RT 45
SETPC 3
SETBG 0
FD 70
RT 90
SETPC 4
SETBG 1
FD 70
LT 135
SETPC 5
FD 50
```

Experiment with different screen and pen colors and you will find that all combinations do not work well together. Some of them wash out each other, and some produce wierd colors when put together. It is not unusual for this to occur, and you must find the combinations that work on your particular monitor and enhance the graphic you have designed. On some monitors a certain pen color may not show up at all on a certain background. If the colors do not even resemble the colors named, adjust the dials on the set until they are more realistic.

Practice Activities

1. Draw a simple graphic like a square or triangle. Enter it several times, using each pen color and background color from 0 to 5. Record which ones look good, which ones distort the color, and which ones do not even display the graphic.
2. Draw the same house that you drew earlier but use the pen color and background color that you like the best from number 1.

Black and White Monitors

Not having a color monitor to use with Logo is a disadvantage but perhaps just more stimulating and challenging. Even though colors are not available, the screen still reacts to the call for color with certain patterns. It is interesting to explore this phenomenon by experimentation.

Some colors will produce black lines, some white lines, and some double lines. The practice activities that follow will allow you to discover some of the peculiarities occurring when using color commands on a black and white monitor.

Practice Activity

Design a simple object or use one you have already written. Enter a background color and pen color with each different possible combination of colors and take notes regarding the resulting effects. Keep these handy for later when you may be able to utilize them for special effects.

3 Developing a Procedure

More Screen Modes

SPLITSCREEN or CTRL S

SPLITSCREEN (or CTRL S) is the most common screen mode. With this command the turtle is shown, and the question mark and cursor are displayed on the fourth line from the bottom left of the screen. The top 20 lines are alotted to graphics, and the bottom four will display the text you enter. If a command is given which causes the turtle to move into the lower portion of the screen, the turtle or line cannot be seen unless the FULLSCREEN command is given. Enter

```
CTRL S
CS
BACK 100
```

FULLSCREEN or CTRL L

Calling for the FULLSCREEN (or CTRL L) mode causes the entire screen to be available as a graphics screen. However, if you make any errors in typing, the screen will revert automatically to a SPLITSCREEN in order to give an error message. It will remain in the SPLITSCREEN mode until the FULLSCREEN command is given again. Otherwise you will not see the instructions as you enter them. Enter CTRL L now. Enter CTRL S.

TEXTSCREEN or CTRL T

TEXTSCREEN (or CTRL T) is exactly the opposite of the FULLSCREEN mode. The entire screen is devoted to text. It will remain in this mode until a different screen or turtle command is given. Enter CTRL T and the screen shows all of your instructions. Enter

```
CS
FULLSCREEN or CTRL L      (Screen shows no text)
FD 300                    (Turtle moves upward and
                          wraps around, coming up
                          from the bottom to com-
                          plete its 300 steps)
SPLITSCREEN or CTRL S
```

Now there are three lines of text at the bottom, plus the ? and the cursor. The lower portion of the graphic line is no longer visible. Enter

```
TEXTSCREEN or CTRL T
```

The graphics are gone now, and all your commands are shown. Enter FD 300.

CLEARTEXT

By using CLEARTEXT (no short form), whatever text is on the screen is erased. This mode does not affect the graphics on the screen, if any. Enter CLEARTEXT. If you enter TEXTSCREEN after using CLEARTEXT, the graphics that you have been working on and the text are also erased.

If you enter SPLITSCREEN after using CLEARTEXT, you will see all the lines that you have been drawing. Enter:

```
CS
CTRL S
RT 45
BK 140
LT 90
FD 60
RT 90
FD 200
CLEARTEXT
```

Use this same program but change the second line to

```
CTRL L
```

The last line has no effect since there is no text on the screen. Leave this design on the screen.

HOME

HOME is an extremely useful command. HOME is the name of the original position of the turtle—in the center of the screen facing up. At any time, if you want the turtle to return to its home position, just enter the word HOME. Everything on the screen will remain as is except for the turtle's location, plus the turtle leaves a trail from where it was to its home.

Leave the graphic on the screen from the previous lines and enter the command HOME to see what the turtle does.

Return to a SPLITSCREEN by typing CTRL S.

PROCEDURES

Up to this point the turtle has been used in the immediate mode—that is, each line entered is executed as soon as the RETURN key is pressed. There has been no way to see the creation again without reentering the instructions. Not to worry!

If you design a spectacular object and wish to see it again, a procedure will place your program in the memory of the computer. The name PROCEDURE is the name for a program that has been placed in the computer's memory. If the procedure is written according to the directions that follow, the computer will remember it until you ask for it to be drawn and then will produce it for you.

In order to commit a program to the computer's memory, there are three steps to follow:

TO

1. The word TO is the keyword for procedures and must be entered first. The infinitive is used because, to the computer, what follows will be a verb—that is, some operation it is going to perform. If a procedure is named STAR, when STAR is called for the computer "stars" by drawing a star. Following the word TO, there must be a space and an original name (never use the same name twice). Here is how the first line would look for a program named LINE.

```
TO LINE
```

There must be a space between TO and the procedure name.

Now press RETURN. This signals the computer that a series of instructions are going to be entered which will be named LINE. Once even one instruction is entered, there is a procedure by that name.

After typing TO LINE and pressing RETURN, the question mark that has normally been at the beginning of each line will no longer be present. Instead, an arrow will appear. This is to signify that you are in the procedure mode as opposed to the turtle (or immediate) mode.

Instructions

2. After the procedure has been named, type in the lines that compose the design. Since you cannot see the turtle in this mode, you should have already tried the program with the turtle in sight (the turtle mode) and have copied the program down. It is not possible to be in the procedure mode and the turtle mode at the same time. Now enter these lines:

```
TO LINE
CS
FD 150
END
```

END

3. It is necessary to include the command END after entering the complete procedure, no matter how many instructions it has. Once END is entered, a message will appear on the screen that says

```
LINE DEFINED
```

You may now enter the word LINE (or whatever the procedure was named). Press RETURN and the procedure will be

produced on the screen. The complete procedure is

```
TO LINE
CS
FD 150
END
LINE DEFINED        (Message on the screen)
LINE                (User input)
```

After the message LINE DEFINED appears, simply type LINE. Type it several times and you will get lines on top of lines.

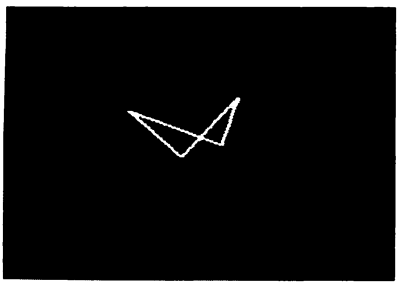
Practice Activities

1. Write a procedure resembling an arrow. Insert a CLEARSCREEN command into the program so that each time you call it, the screen is cleared first.
2. Write a procedure to draw a triangle. Use the PENUP and PENDOWN commands so that the triangle is not drawn in the center of the screen.

More on Procedures

Following is a program to draw a butterfly. The instructions are written in a procedure named BUTTERFLY. Enter it into the computer as shown:

```
TO BUTTERFLY
RT 45
FD 65
LT 25
BK 40
LT 90
FD 80
LT 160
FD 55
HT
END
```

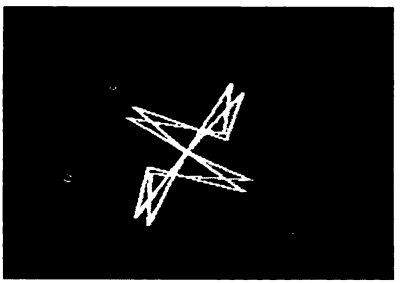


The message, BUTTERFLY DEFINED, signals that the procedure was entered correctly and is now in the computer's memory. Type BUTTERFLY and it will be produced on the screen. Type it again and again to make a design of butterflies. The screen will not clear each time unless you insert the CLEARSCREEN command as the first line of the procedure. Inserting instructions and changing procedures is done by editing, which will be presented in Chapter 6. If there is an error in the procedure, this will be indicated only after the procedure is run.

Procedures within Procedures

It is possible to produce the same effect as the butterfly design above in this manner:

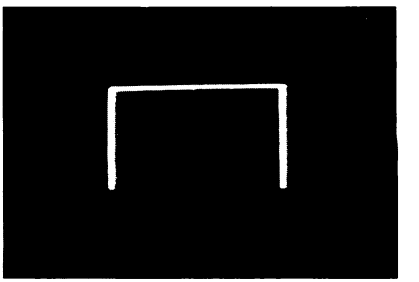
```
TO BUTTERFLIES
BUTTERFLY
BUTTERFLY
BUTTERFLY
BUTTERFLY
END
```



The procedure BUTTERFLY is called for in the procedure named BUTTERFLIES. When you enter the name BUTTERFLIES, four of the BUTTERFLY procedures are reproduced and the program stops. Enter BUTTERFLIES again and the turtle retraces its steps.

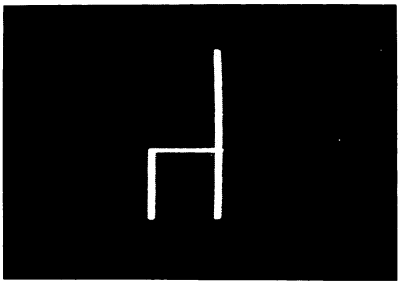
Another example of procedures within procedures is when you want some different objects to be on the screen but want to design them separately. Perhaps you want to create TABLE, CHAIR, and PANE procedures. First, enter each procedure separately:

```
CS
TO TABLE
PU
BK 20
PD
FD 60
LT 90
FD 100
LT 90
FD 60
END
```



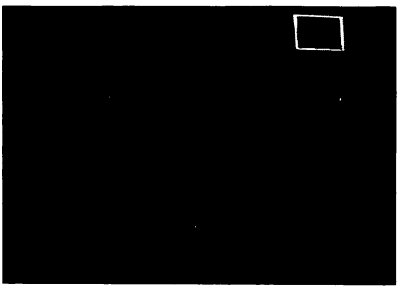
```
TO CHAIR
PU
BK 50
RT 90
FD 20
LT 90
```

```
PD
FD 40
RT 90
FD 40
RT 90
FD 40
BK 100
END
```



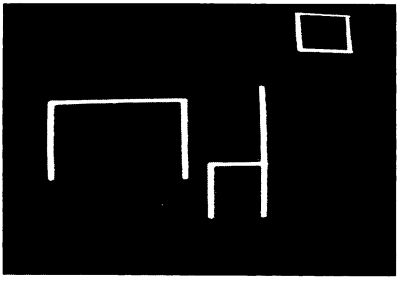
Note: The name WINDOW cannot be used to name a procedure. It is already defined as a primitive. Therefore, name the procedure PANE.

```
TO PANE
PU
FD 80
LT 90
FD 150
PD
FD 40
RT 90
FD 30
RT 90
FD 40
RT 90
FD 30
HT
END
```



Now that your procedures are defined, you can write one to combine them all:

```
TO ROOM
CS
TABLE
PU
HOME
PD
CHAIR
PU
HOME
PD
PANE
END
```



Type the procedure named ROOM. The three separate procedures called for in ROOM are produced.

Why were the PENUP and PENDOWN commands necessary? Of course it is because we don't want to leave a trail as we position the turtle for drawing.

Why were the HOME commands inserted in this procedure? Notice that each procedure was written assuming the turtle to be in the center of the screen. Without the HOME commands in ROOM, the objects would not be in controlled positions. You can check this by entering the following:

```
TABLE
CHAIR
PANE
```

Do you see the significance of the HOME command in this context?

PRINTOUT Procedures

POPS

Once the graphic is drawn, the instructions can no longer be seen. Now type CTRL T. If you wanted to go back and check your procedures, type POPS, which stands for PRINTOUT PROCEDURES. All the procedures that have been entered at this sitting will be shown on the screen. They will probably take up more than one entire screen. If this occurs, the text screen will *scroll* the lines and stop when it gets to the end. If you want to see the beginning part of the text, CTRL W is used (WAIT). By typing any other character, scrolling will resume. The CTRL W can be used at any time you wish to temporarily stop what is occurring on the screen. Remember that to continue the scrolling press any key. If the procedure you wanted to look at went by too fast, enter POPS again and be ready with the CTRL W.

POTS

If you just wanted to see what procedure names have been used, simply enter the command POTS for PRINTOUT TITLES. All the procedure titles you have entered will be displayed. If there are so many that they also scroll (this is doubtful but possible), the CTRL W can be used again.

PO

One last command utilized for printing out information on procedures is PO, which stands for PRINTOUT.

```
PO "CHAIR
```

This command lists instructions for the CHAIR procedure. It can also be used with more than one name:

```
PO [TABLE CHAIR]
```

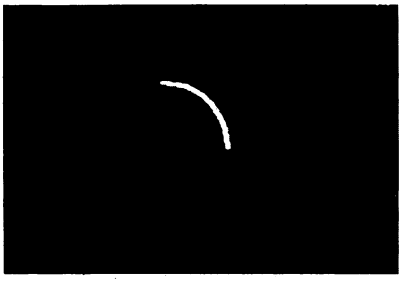
In the second example the PO command is followed by a space, a left bracket, the procedure names separated by a

space (there can be any number of names), and a right bracket. The left bracket is produced by holding down the SHIFT key and typing the letter N. The right bracket is produced with a SHIFT M. Some Logo materials come with key covers to place over the N and M since it is a specialized use of the keys. Parentheses can *not* be substituted.

Stored Procedures

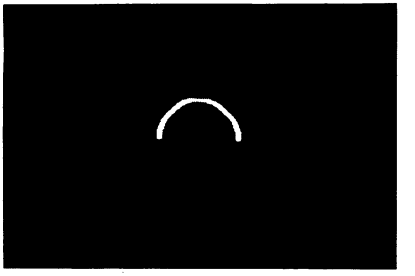
There are some procedures which have been included on the Logo disk for your use. When you load Logo into the computer, these prewritten programs are automatically loaded as well. The first procedure in the library is named ARCLEFT (or ARCL for short). As you might guess, it is a procedure to produce an arc to the left of centerscreen. To use this procedure, it must be followed by two numbers; the first represents the size of the radius in turtle steps; the second represents the number of degrees you desire. For example,

```
ARCLEFT 50 90        or        ARCL 50 90
```



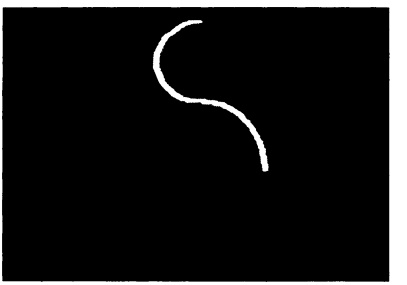
will produce an arc with a radius of 50, and 90 degrees in length—which is 1/4 of a circle (since 4 times 90 = 360). The arc begins where the turtle is and in the direction it is headed. To draw an arc to the right with a radius of 30 that is half a circle, enter

```
CS
ARCRIGHT 30 180      or        ARCR 30 180
```



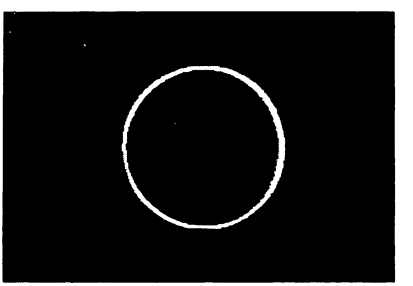
Carefully planned, curves can be combined to make some interesting designs. This time clear the screen and enter both lines

```
ARCL 50 90
ARCR 30 180
```

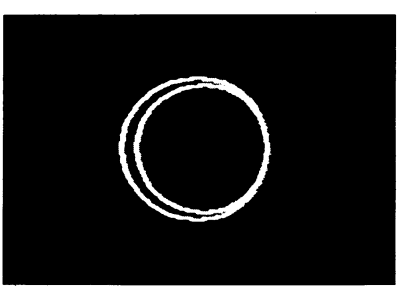


Two more library procedures are CIRCLEL and CIRCLER, for circle left and circle right. Only one number or input is necessary with these procedures, and that is the radius. Why is it not necessary to include the degrees? The procedure is already programmed for 360 degrees so the only missing information is the radius.

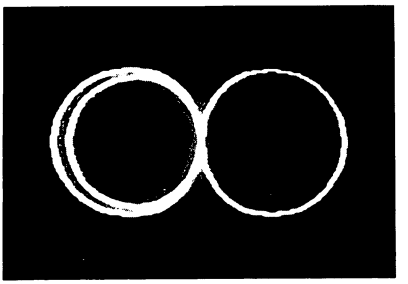
```
CS
CIRCLEL 50
```



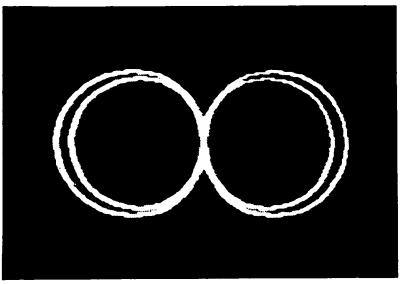
```
CIRCLEL 45
```



CIRCLER 50



CIRCLER 45

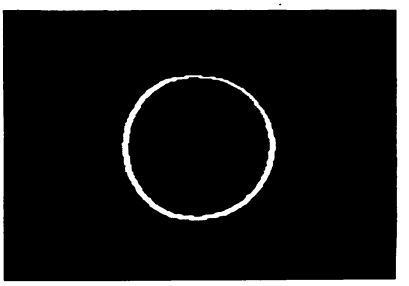


These four lines will draw a circle inside a circle on the left and the same on the right. What is the diameter of the large circles? Since the radius is half the diameter, the diameter would be 2 times 50, or 100.

Using these procedures will spoil you when you are asked to draw circles from scratch later. Can you logically describe how circles are made?

Two more library procedures are ARCR1 and ARCL1. The two inputs for these do not include the radius. The first number represents turtle steps, and the second represents the number of times you want the turtle to move that number of steps. The turtle automatically turns 10 degrees after each series of steps. For example, the command

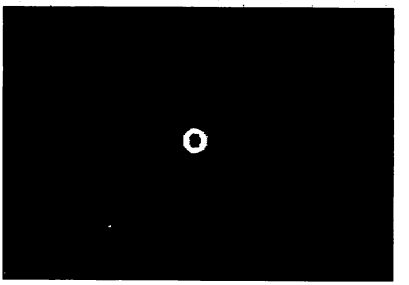
ARCR1 10 36



instructs the turtle to move 10 steps, 36 times. Since it turns 10 degrees each time, the line should produce a large circle,

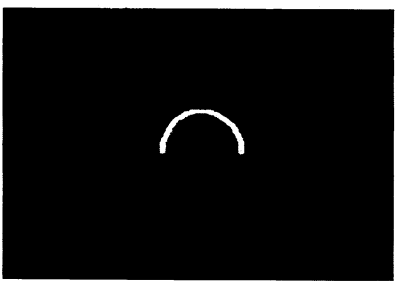
each section being 10 steps long. What size will the circle be? What will this line produce?

```
ARCL1 1 36
```



Was your prediction correct? It is a tiny circle. Here's another one to visualize:

```
ARCR1 5 18
```



This will be a medium-sized, half-circle, going a total of 180 degrees (18 times 10 degrees).

Practice Activities

1. Draw five circles of different sizes on the right of the screen, using CIRCLER; do the same using ARCR1.
2. Draw arcs to the right and left that keep getting larger and longer.
3. On the left, draw a small circle, a large arc, and a circle using ARCL1. Produce a mirror image on the right.

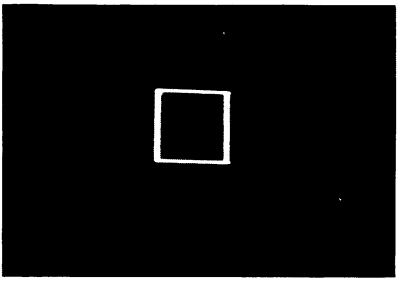
REPEAT

4

Saving Time with REPEAT

Recall that when you wish to write a program that will be drawn each time you enter its name, you need to use the word TO, leave a space, give the program a name, and then enter the instructions. In the program that follows, a box is created.

```
TO BOX
FD 50
RT 90
FD 50
RT 90
FD 50
RT 90
FD 50
RT 90
END
```



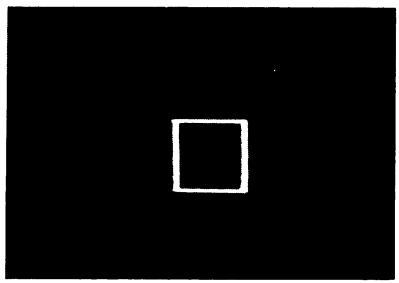
It may occur to you that there must be a way to avoid the repetition in this program. You are right! There are four sets of FD 50 RT 90 instructions. Using a command called REPEAT, the program can be reduced to three lines, instead of 10. For example, let's call the procedure above SQUARE and use REPEAT in the instructions.

REPEAT

In the REPEAT command, three components are necessary. The word REPEAT, the number of times you want something repeated, and the actual set of instructions—enclosed in brackets. The brackets are not shown on the keyboard but are produced by holding down the SHIFT and pressing N (for a left bracket) or M (for the right bracket).

To write an abbreviated procedure of the BOX program, the first line would be

```
TO SQUARE     (A different name is necessary since there is
              already a procedure named BOX if you
              entered the previous procedure)
REPEAT 4 [FD 50 RT 90]
END
```



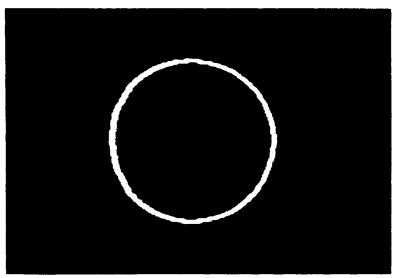
Notice that after the word REPEAT is the number 4. This simply refers to the number of times that you want the instructions in the brackets to be repeated. Inside the brackets are the instructions FD 50 RT 90. The first time this line is read, the turtle will go upward and turn 90 degrees to the right. The next time it will travel 50 steps toward the right side of the screen and make a 90 degree turn so that it is facing down. The third time around it will proceed 50 steps again, turn right (facing the center now), and make its last 90 degree turn, leaving it where it began. The program is now finished.

The instructions in the brackets can be repeated any number of times. If you wanted to draw a circle (you knew it was coming sooner or later), the long form would be to use these two lines 360 times:

```
RT 1
FD 1
```

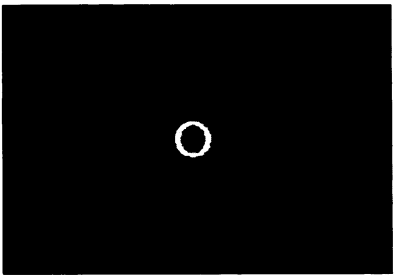
The turtle would turn one degree and move forward one step, 360 times. Putting this into a REPEAT command it would look like this:

```
REPEAT 360 [RT 1 FD 1]
```



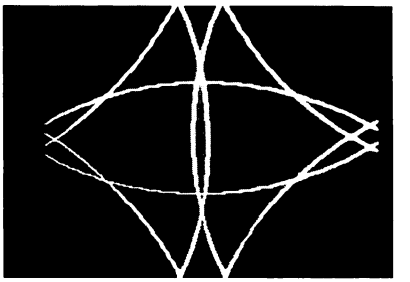
What would the circle look like if the line were

```
REPEAT 72 [RT 5 FD 1]
```



Try this line and let it go until it is finished. Can you figure out what's going on?

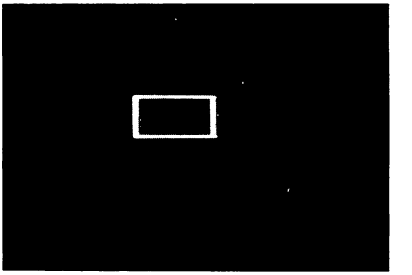
```
REPEAT 360 [RT 1 FD 5]
```



Eventually the circle is completed, but it is so large it wraps all over the screen!

Getting a little fancier, what will this program produce?

```
FD 30
RT 90
FD 60
RT 90
FD 30
RT 90
FD 60
RT 90
```



Another way of writing the same program would be:

```
REPEAT 2 [FD 30 RT 90 FD 60 RT 90]
```

In this example, two sets of four instructions were repeated. Any time more than one set of instructions follow each other in a program, the REPEAT command can be used.

Practice Activities

1. Draw a very small square and a very large square.
2. Draw a circle as large as possible without going off the screen.
3. Draw three circles of different sizes inside each other.
4. Draw a triangle with equal sides.

Repeating Procedures

To review for a minute, look at this procedure:

```
TO LINE
RT 45
FD 150
END
```

With the program named LINE, remember that you could write a procedure like this:

```
TO LINES
LINE
LINE
LINE
LINE
LINE
END
```

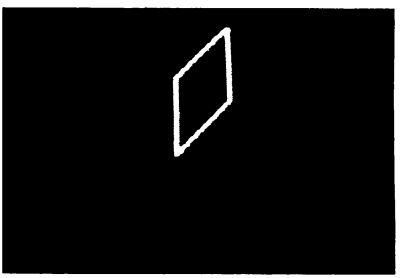
The turtle would produce five lines on the screen. Is it possible to use the REPEAT command in this situation? Of course it is. Compare the following procedure with the one above; their output will be identical.

```
TO LINES
REPEAT 5[LINE]
END
```

Notice that after the word REPEAT is the number 5, the number of times LINE is called for in the first program. In the square brackets is the name of the procedure (LINE) you want repeated. This shortcut comes in handy when you begin to write more procedures and want to insert standard objects (such as SQUARE), in new procedures.

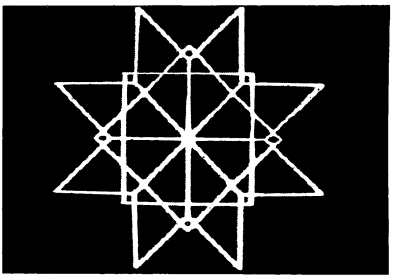
To illustrate, write a procedure named DIAMOND:

```
TO DIAMOND
FD 60
RT 45
FD 60
RT 135
FD 60
RT 45
FD 60
END
```



Using the procedure named SQUARE, you could make an interesting design in this way:

```
FULLSCREEN          or          CTRL L
REPEAT 8[SQUARE DIAMOND]
```



Procedures already defined (like SQUARE and DIAMOND) can be used in other procedures.

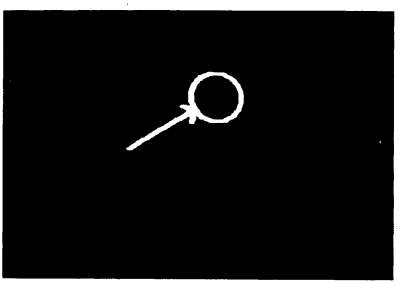
Practice Activities

1. Draw a square using REPEAT.
2. Write a procedure called SQUARES that includes your instructions in Activity No. 1 above. Using REPEAT, draw the square on the screen 4 times.
3. Draw circles all over the screen, manipulating the pen up and down.

Balloon

Using the turtle commands that have been introduced so far and throwing in the REPEAT command, we will attempt to simulate a balloon. Study the REPEAT command to understand exactly what is occurring:

```
TO BALLOON
RT 60
FD 60
RT 90
REPEAT 120[LT 3 FD 1]
RT 45
FD 10
BK 10
RT 90
FD 10
BK 10
RT 135
BK 60
LT 90
END
```



The REPEAT command is responsible for the circle portion of the balloon. Run the program to see what it looks like and then take a look at the procedure again. You can do this by using a command called PRINTOUT (or PO for short), followed by the name of the procedure you want to look at. Type

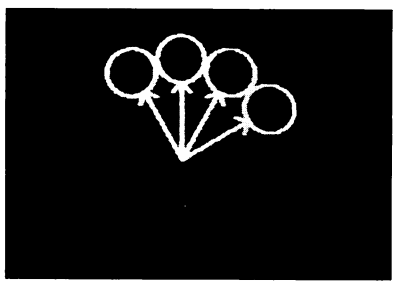
```
PO "BALLOON
```

Now press RETURN. Because the procedure is over four lines long, remember that you can obtain a full text screen by typing CTRL T for TEXTSCREEN. This command will automatically revert the screen to allow you to see a full screen of text.

Balloon Bouquet

Let's put several balloons together in a procedure and make a balloon bouquet. It will look like this:

```
TO BALLOONS
REPEAT 4[BALLOON]
END
```



We begin with the procedure name on the first line as usual. The REPEAT command follows, with the number 4 designating the number of times we want the procedure BALLOON reproduced. The END statement completes the procedure.

Run the program BALLOONS and see what happens. Notice that the balloons are nicely spaced and are all coming from the same spot on the screen. Why is this so? Can you find the lines in the BALLOON program that cause this to happen?

Hint: The first line in the BALLOON program is a right turn of 60 degrees, so we would expect the first balloon to be drawn at a 60 degree angle from center screen. The balloon is then drawn, and the turtle returns to the center but is still facing 60 degrees right. Look at the last line in the program; LT 90. It turns the turtle 90 degrees to the

left—but why? If the turtle is facing 60 degrees right and turns 90 degrees to the left, it will be heading 30 degrees to the left of center—right? So far, so good. When the REPEAT command is executed again, what is the first thing that happens? Line one says turn right 60 degrees. If the turtle has stopped at 30 degrees to the left and turns to the right 60 degrees, how many degrees to the right of center is it facing? The answer is 30.

Following this same process two more times, you will see that each balloon is drawn 30 degrees apart. If these degree changes were not inserted into the procedure, the balloons would be drawn on top of each other four times. It is very challenging to manipulate the turtle's heading and location to get the desired results in procedures.

Practice Activities

1. Write a procedure to make a triangle with equal sides of 50 steps. Use TRIANGLE in a procedure called TRIANGLES so that it is drawn 20 steps apart until it reaches the point where it began.
2. Draw lines coming out from the center point in a circular design. Make them approximately 10 degrees apart.
3. Draw horizontal lines across the screen, from left to right, 25 steps apart, from bottom to top.

5 Loading and Saving Programs

No doubt you will want to keep the many creative programs you design with Logo. You can prepare (also referred to as formatting or initializing) a blank disk for this purpose. Blank disks are available at all computer stores for less than $5.00. Each blank disk is put through the initialization process *only* once. If you attempt to initialize a disk already initialized, you will lose whatever has already been saved on the disk. Here is how it's done:

1. Turn the computer off. The switch is on the back, left-hand side. There is a square on the keyboard labeled POWER which is lit up when the power is on.
2. Insert the Apple 3.3 DOS System Master into the disk drive and turn on the power switch.
3. The small red light on the disk drive will go on and you will be able to hear the drive working. Once the light on the drive goes off, remove the System Master disk, insert your blank disk, and type

   INIT HELLO (Press RETURN)

   The disk drive will run for about two minutes to complete the initialization process.
4. Check to see if you were successful in initializing your disk by typing

   CATALOG (Press RETURN)

   The CATALOG is a listing of all files that you have on a disk. You can type CATALOG and check your files at any time.
5. The screen should show your HELLO file on the screen.
6. Remove your newly initialized disk from the drive, turn off the power, insert the Logo disk, and press RETURN.
7. On the screen you will see

   PRESS THE RETURN KEY TO BEGIN
   IF YOU HAVE YOUR OWN FILE DISKETTE,
   INSERT IT NOW, THEN PRESS RETURN

   Do not insert your own disk at this time. Press RETURN and the Logo STARTUP file will be loaded into the computer. This is the series of prewritten files with

which you have already worked (including circles and arcs).

8. When you see the following message displayed

```
WELCOME TO LOGO
?
```

take the Logo disk out of the drive and insert the initialized disk.

9. Type the following:

```
SAVE "STARTUP "AIDS
```

10. Again you will receive this message:

```
WELCOME TO LOGO
?
```

Type and return CATALOG and you should see the HELLO and STARTUP programs on the menu.

11. From now on, after you load Logo into the computer, insert your initialized disk when you receive the following screen message

```
IF YOU HAVE YOUR OWN FILE DISKETTE,
INSERT IT NOW, THEN PRESS RETURN
```

12. The initializing procedure is now complete. *Remember*, this process need only be followed the first time you format each blank disk.

Saving Procedures

Saving programs is a bit confusing if you do not read this section carefully. There are some peculiarities which you need to be aware of. For example, right after Logo and your initialized disk have been loaded, let's say that you write a procedure named BIRD. You could type

```
SAVE "BIRD
```

You would receive the message

```
1 PROCEDURES SAVED
```

No problem! However, next you write a procedure named FLOWER. If you save it as you did BIRD (SAVE "FLOWER), you will receive the message

```
2 PROCEDURES SAVED
```

If you then wrote a procedure named TREE and saved it (SAVE "TREE), guess what message would appear?

```
3 PROCEDURES SAVED
```

Do you see that each time you save one procedure, every

other procedure you have written to that point is accumulated under the new file name? This is extremely inefficient and undesirable.

One way to avoid this problem is to write the three procedures and then save them under a collective name like SCENE or BIRDFLOWERTREE. (File names can have up to 32 characters.) The line would look like this:

```
SAVE "SCENE
3 PROCEDURES SAVED      (Message on screen)
```

If you continued writing more procedures, named TABLE, CHAIR, and LAMP, and typed SAVE "ROOM, the screen output would be

```
6 PROCEDURES SAVED      (It accumulated them all
                         again!)
```

The procedures saved in ROOM will be BIRD, FLOWER, TREE, TABLE, CHAIR, and LAMP. To avoid this you could turn off the computer, thus erasing all procedures from memory (except those saved on your disk) and begin writing procedures again. However, this is time consuming and not very practical. If you sit down to write several procedures and are satisfied to save them *all* at the end of the sitting under one collective name, then that is fine. However, there are two ways of getting around this problem.

Some special comments are appropriate first. Never use the same procedure name twice. If you do, you will create problems for yourself. Making a record of the procedure names you use if necessary or checking your files with the POTS command if you are in doubt will help you avoid this problem.

Saving Procedures in Packages

In order to save the desired procedures in specific file names, the PACKAGE command will do the trick. Let's say that six procedures named BIRD, FLOWER, TREE, TABLE, CHAIR, and LAMP are in the computer. You want the first three in one file and the other three in a second file. Here are the commands:

```
PACKAGE "SCENE [BIRD FLOWER TREE]
SAVE "SCENE "SCENE
3 PROCEDURES SAVED      (Message on screen)
PACKAGE "ROOM [TABLE CHAIR LAMP]
SAVE "ROOM "ROOM
3 PROCEDURES SAVED      (Message on screen)
```

In the first line above, the first three procedures (BIRD, FLOWER, and TREE) are placed inside a special *package*

which is exclusive of anything other than those procedures named inside the brackets. The file name, SCENE, is preceded by a space and quotation mark. After the package name and inside brackets (SHIFT N and M) are the procedures you want placed in the package.

In the SAVE command in which the package is saved (line two), it is necessary to name the file again (the first SCENE) and then name the package in which the selected procedures were placed. The file name (first) could be different from the package name (second), but it is logical to use the same name.

The literal translation of the first SAVE command listed above is, "Save, in a file named SCENE, whatever is in the package named SCENE."

Lines 4 and 5 repeat the process for three different procedures.

If you later forget what procedures you have packaged and saved in a specific file name, simply load the file (loading instructions are in the next section) and use the POTS command for a listing of all the titles. For example,

```
POTS "SCENE      (User input)
TO BIRD
TO FLOWER        (Screen output)
TO TREE
```

ERASE and ERASEALL

A second way to keep from accumulating all your procedures in each subsequent file is to erase every procedure that you don't want to be saved. The command ERASE (ER is the short form) must be followed by a space, quotation mark, and one procedure name. If more than one procedure needs to be erased, place them in brackets. Each procedure you wish to eliminate from the workspace, to get out of your way, must be listed. If there were five procedures in memory, TABLE, CHAIR, LAMP, FLOWER, and TREE, and you wanted to save the first three, the command would look like this

```
ER "FLOWER
ER "TREE
```

or like this

```
ER [FLOWER TREE]
```

You could then write a procedure to save the others in a file named ROOM:

```
SAVE "ROOM
3 PROCEDURES SAVED
```

If you wished to save all the procedures except TREE, you could write the command

```
ER "TREE
```

Note: Any time just one word or name is going to be called, it is acceptable to use a quotation mark preceding the name. If more than one procedure is to be named, they must appear in brackets separated by spaces.

The ERALL command wipes out everything from the memory, even those procedures in packages. However, nothing that has been *saved* can be erased with the ERALL command.

Let's look at how the ERALL command can be used with package names. If you have a package named ROOM with TABLE, CHAIR, and LAMP in it and you give the command

```
ERALL "ROOM
```

The command will only affect those procedures in that package, and the others in the workspace will remain. The two ERASE commands (ERASE and ERALL) only affect what is in the workspace or memory at the time, not what is on the disk.

Is It Really Saved?

At this time if you turn off your Apple and come back later and load Logo and your save disk, you will probably want to check to see if your procedures really have been saved. There are two ways of checking for saved programs:

CATALOG

After receiving the message WELCOME TO LOGO, type in the word CATALOG and press the RETURN key. This requests that all the files saved on the disk be listed on the screen. Look for the programs you have saved to be on the list. If they are not there, the files were not saved correctly.

LOAD

The word LOAD followed by a space, quotation mark, and the name of the file you saved, will place the named file into the computer's memory or workspace. Until a file is loaded, it cannot be run. If the message FILE NOT FOUND appears, the file has not been saved. It will also not appear if you type CATALOG. Go back and review the saving procedures until there are successful results.

Loading Procedures That Have Been Saved

After loading Logo and the initialized disk with saved programs stored on it, enter

```
CATALOG
```

This is optional and is only necessary if you want to check what files have been saved on the disk. Access any program with a LOAD command followed by a space, quotation mark, and the name of a file you have saved.

```
LOAD "SCENE
```

You will receive an error message if you attempt to run a procedure that has not been loaded.

After entering the LOAD command, the disk drive will go on. When the disk light goes off, the ? appears, signaling that the procedures in SCENE are now in memory. Type BIRD (or FLOWER or TREE) and the design will be drawn. By loading SCENE, all the procedures saved in that file are also loaded.

Enter the file name SCENE and you will receive the following error message

```
I DON'T KNOW HOW TO SCENE
```

This is because there is not a *procedure* named SCENE. It is *only* the name of a file that contains the procedures BIRD, FLOWER, and TREE.

POPS, POTS, PO

Some additional uses of these three commands that have already been introduced will be helpful. The POPS command, when given without any input,

```
POPS
```

will list each procedure name and contents in the workspace. It can be used with package names also:

```
POPS "ROOM
```

This will list names and contents of the procedures in the package named ROOM.

POTS is similar to POPS except that it only prints out the *names* of the procedures in the *package* and not the instructions associated with each. When used with no inputs, it lists all the procedures in memory. The package name can also be specified:

```
POTS "ROOM
```

PO, the PRINTOUT command, is not used with packages or files but only with procedure names. It will print out the names and contents of each procedure named. It must have a procedure name as input. If several procedures need to be examined, they can be listed inside brackets:

```
PO [FLOWER TREE]    or    PO "FLOWER
```

Practice Activities

1. Write a program for TABLE, CHAIR, and LAMP. Save the procedures in a file named ROOM. Turn off the computer, reload, and check to see if they were actually saved.
2. Write procedures for TREE and FLOWER. Place them in a package named SCENE, and save them in a file named SCENE.
3. Load all five procedures into memory. Print out the procedures in the file ROOM. Print out the procedures in the package SCENE.
4. Erase the procedures in the SCENE package from memory. Use POTS to check if you were successful.
5. Erase the procedures left in the workspace. Check to see if you were successful.

Editing

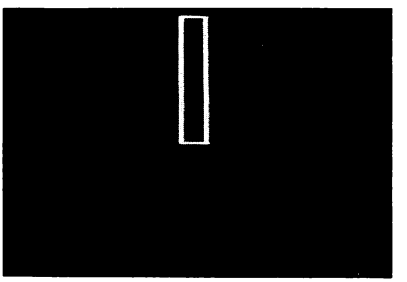
6

The Edit Mode

An extremely useful method of making changes in your procedures is the *editor*. In this mode, you can easily make adjustments in a procedure you have already defined. In other words, this gives you the chance to make corrections, additions, deletions, or any kind of alterations to procedures you have created.

Enter the following procedure, typing ERALL first:

```
TO POST
FD 100
RT 90
FD 20
RT 90
FD 100
RT 90
FD 20
RT 90
END
```



Run POST. It should resemble a tall, narrow rectangle. To edit the procedure called POST, the EDIT command is

```
EDIT "POST   or   ED "POST
```

Enter one of the above commands; they have identical results. After pressing RETURN, immediately the screen will list the procedure labeled POST (or whatever program name you want to edit). Across the bottom of the screen will be a white band with the words LOGO EDITOR. This indicates that you are now in the edit mode and are able to make the desired changes. In order to do so, special control characters must be used.

Control Characters

The edit mode is utilized by a series of control characters. For example, by holding down the key marked CTRL and pressing the E (for end), the cursor will automatically move to the end of the line it is currently on. Each control character has a different duty.

- **CTRL A:** No matter where the cursor is, this places it on the *first letter* of the line it is on.
- **CTRL B:** This command has the power to move the cursor *back* one character at a time without erasing the

ones it is passing over. It does not have the same effect as the left arrow (←), which does erase each character it passes over.

- **CTRL C:** A key command, the CTRL C is used to get you out of the edit mode and writes your changes into memory. It will not automatically add the END command if you happened to forget it. After pressing the CTRL C and pressing RETURN, the screen will say (if the procedure was named ONE):

  ```
  ONE DEFINED
  ?
  ```

- **CTRL D:** The character under the cursor is *deleted.*
- **CTRL E:** This command moves the cursor to the *end* of the line it is on
- **CTRL F:** Use CTRL F to move the cursor *forward* one position on the line without erasing. This can be used interchangeably with the right arrow (→).
- **CTRL G:** This is the opposite of CTRL C. If you use this command you will be removed from the edit mode *without* your changes being written into memory. All changes are *ignored.*
- **CTRL K:** Noting the cursor position, all characters from it to the end of the line are *killed* or deleted. An interesting thing happens here; whatever information you killed is placed in a holding space so that you can reinsert it at another position. See CTRL Y.
- **CTRL L:** This command is useful when you have a long procedure. It will move the line on which the cursor appears to the middle of the screen, showing what comes before and after that line. This only works if the cursor is more than halfway down the first screen full of instructions.
- **CTRL N:** A much used control, CTRL N makes the cursor move to the *next* line.
- **CTRL O:** A blank line on the line just above the cursor is *opened* up for additional lines to be added when the cursor is on the first character of the line. If the cursor is in the space at the end of the line, the open line is inserted below the cursor.
- **CTRL P:** CTRL P moves the cursor to the *previous* line. It is a frequently used control.

- **CTRL V:** This command causes the next "page" of text to appear. If your procedure is quite long, you would use this to get to other parts of it quickly.
- **CTRL Y:** Used in conjunction with CTRL K, it reinserts the line you killed into the position where the cursor is located. Usually you would place the cursor on a blank line before using this command.
- **RETURN:** After a line has been edited, it will be saved with the changes by pressing RETURN.

Here are several changes to make to the POST procedure. Make all the changes in the edit mode. Use the control keys explained above to make all the additions, deletions, and replacements necessary. Notice that even the name of the procedure is being changed. That means that after you edit it, you will still have the old version under the name POST, plus the new, edited procedure named FATPOST.

1. Enter the edit mode if you are not already in it.
2. Move the cursor to the end of the title line.
3. Move the cursor to place it on the P of POST without erasing anything.
4. Enter the new name FATPOST.
5. Delete the name POST.
6. Move the cursor to line four (FD 20).
7. Insert a blank line after line four.
8. Move the cursor to the beginning of line three: RT 90.
9. Move the cursor to the end of line two.
10. Place the cursor on the 1 in 100 on line two without erasing anything.
11. Move the cursor to the beginning of line two without using CTRL B.
12. Kill line eight (FD 20) and reinsert it in the blank line you created. (**Hint:** Two control procedures must be used—K and Y.)
13. Add this in the blank you just created: FD 40.
14. Get out of the editor mode in a way that will record all the changes just made.

15. Run the POST procedure and then the FATPOST procedure. There are now two different procedures.
16. Get back into the editor mode with the FATPOST procedure and change the name to POST2. Use the control character to exit the editor which will *not* record this change. Check to see if you were successful by typing POST2. Since CTRL G ignores any changes, you will receive the error message

    I DON'T KNOW HOW TO POST2

CTRL L and CTRL V were not used because they are for long procedures. There will be an example of this later.

It will be helpful to you to commit the control commands to memory. It will automatically happen as you use them and will save you much time with your editing.

Saving Your Edited Procedure

Place these two procedures in one package named POSTS with the command:

```
PACKAGE "POSTS [POST FATPOST]
```

Now save the package by entering the command:

```
SAVE "POSTS "POSTS
```

You will receive the message:

```
2 PROCEDURES SAVED
```

If you type CATALOG at this time, you will see that the procedure POSTS has been saved.

Enter POST and let it run, then enter FATPOST and let it run. The two procedures are now saved. Enter POSTS and you again receive an error message. Why? Because there is no procedure defined as POSTS, just a package and a file named POSTS containing the two procedures POST and FATPOST.

Editing Saved Procedures

You may decide to change a procedure that has already been saved. Assuming you have only saved the procedure once (that is, in one package or by itself), here are the steps to follow:

1. Load the procedure or the package including the procedure. We are going to edit FATPOST which is in POSTS.

   ```
   LOAD "POSTS
   ```

2. Place the procedure you wish to change in the editor:

   ```
   ED "FATPOST
   ```

3. Edit the procedure by eliminating the last three lines.

4. Exit the editor with CTRL C and look for the message

   FATPOST DEFINED

5. The old file POSTS must be erased in its current form because the wrong FATPOST procedure is saved in it.

   ERASEFILE "POSTS

6. Replace POST and FATPOST (the edited version) in a package:

   PACKAGE "POSTS [POST FATPOST]

Note: What would happen if you skipped this step and just typed SAVE "POSTS? (All the procedures up to that point would be saved in the POSTS file.)

7. Save the two procedures once again:

   SAVE "POSTS "POSTS

8. Check the procedures by typing FATPOST to see if it has been changed with the omission of the last three lines. The procedure POST remains unchanged.

This may seem like a tedious process to go through, and you are absolutely correct. However, it becomes easier with practice.

Editing Procedures Saved More Than Once

Following the same logic that we have gone through with *save* and *package*, here is some information on editing procedures that have been saved in more than one file.

1. Let's say that you have a procedure named TRIANGLE in three files. One file is named TRIANGLES; one is named TREE; and the third is named SHAPES.
2. You want to change the TRIANGLE saved in the files named TRIANGLES and SHAPES, but not the one in TREE.
3. Load TRIANGLES, call TRIANGLE to the editor, and make the changes. Use the commands

   ERASEFILE "TRIANGLES
   SAVE "TRIANGLES

   The edited version of TRIANGLE is now in the TRIANGLES file.

4. Load SHAPES, call the TRIANGLE procedure into the editor, and make any changes you wish. Use the ERASEFILE and SAVE commands as you did in number 3 (except for names):

   ERASEFILE "SHAPES
   SAVE "SHAPES

5. Since you do not desire any changes in the TRIANGLE design in the file named TREE, simply leave it as it is.
6. It is conceivable you could have several procedures with the same name in different files that are all different because of this editing procedure.
7. **CAUTION:** Remember that each time you save anything, if it is not the only procedure in memory, each procedure will be saved. Extensive use of packaging is the effective path to follow.

Deleting Procedures from the Disk

In order to completely delete a file from the disk, use the command

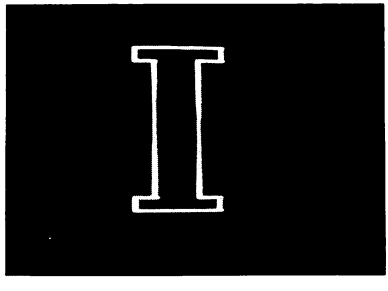
```
ERASEFILE "POSTS
```

POSTS will not show up if you now type CATALOG.

Practice Activities

1. Enter the program of the number 1 on page 10 and save it in a package named ONE. Then edit it to resemble a Roman numeral I, listed below. Do not just enter the I procedure since this is a practice in editing.

```
TO I
BK 40
FD 100
LT 90
FD 15
RT 90
FD 10
RT 90
FD 65
RT 90
FD 10
RT 90
FD 17
LT 90
FD 100
LT 90
FD 15
RT 90
FD 10
RT 90
FD 63
RT 90
FD 10
RT 90
FD 16
HT
END
```



2. Save this procedure in a package named I.
3. Save both procedures in a package named ONEI.
4. Edit the I procedure adding a pen color of blue to the procedure and save it in a package called BLUEI.
5. Check each step by printing out and running all procedures to see if you completed the activity successfully.

7 Recursive and Coordinates

A Little Magic

There is a command which erases the screen but leaves the turtle where it ended up. This is the exact opposite of which other command? The HOME command returns the turtle to center screen but leaves a trail. The command CLEAN allows some interesting effects. When used, everything on the screen disappears! For example:

```
FD 50
LT 45
FD 25
CLEAN     (Only the turtle remains on the screen)
RT 90
FD 25
RT 135
FD 100
CLEAN
```

By following these steps you can see what CLEAN is doing. Let's try a more complicated program (and add some color for fun) and put it in the form of a procedure:

```
TO MAGICSQUARE1
FULLSCREEN
SETBG 5
SETPC 1
RT 90
FD 100
CLEAN     (This command does not affect colors)

RT 90
FD 100
CLEAN
RT 90
FD 100
CLEAN
RT 90
FD 100
CLEAN
MAGICSQUARE1
END
```

An important item to remember is that after using CLEAN the turtle remains where it is after the screen is erased. How could you write this procedure more efficiently? Enter and run your program and see if it works. It might look something like this:

```
TO MAGICSQUARE2
FULLSCREEN
SETBG 5
SETPC 1
RT 90
FD 100
CLEAN
MAGICSQUARE 2
END
```

Recursive

There seems to be a little problem. The program won't stop running. Ignore it for a minute and read on. When you call a procedure within the procedure by the same name, this is called a *recursive* command. Each time the procedure name is encountered in the procedure, the run of the program goes back to the beginning, follows the directions within the procedure, then comes to the procedure name again, and so forth. This procedure will continue to run forever, or until you stop it. There are three ways to stop a procedure from being executed (one of which you have previously been introduced to):

CTRL G

CTRL G causes the program to stop and puts you back in the turtle mode. (G could stand for *give up*. It's the only word I could think of, but you can probably come up with one that makes sense to you.) Now type CTRL G to stop the program.

CTRL W

CTRL W causes the program to *wait* or pause. Upon any key being pressed, the program is continued from the point at which it paused. Run the program again using the CTRL W to cause the execution to temporarily pause, then press any key to make it continue.

CTRL Z

CTRL Z also causes the program to pause. By typing CO for continue, execution is resumed. Try out the CTRL Z and CO commands.

Stop the program using CTRL G and clear the screen.

Here is a program that is a bit more sophisticated than the last one.

```
TO MAGICSTAR
FULLSCREEN
HT
SETPC 3
SETBG 4
FD 100
CLEAN
```

```
HOME
RT 45
FD 100
CLEAN
HOME
RT 90
FD 100
CLEAN
HOME
RT 135
FD 100
CLEAN
HOME
RT 180
FD 100
CLEAN
HOME
RT 225
FD 100
CLEAN
HOME
RT 270
FD 100
CLEAN
HOME
RT 315
FD 100
CLEAN
HOME
CS
MAGICSTAR
END
```

You may be saying to yourself something about the repetition of this procedure. If so, you are on the right track. There is an easier way which will be covered in Chapter 9. For now, enter it this way so that you understand each line.

Again a recursive command has been used; the procedure MAGICSTAR is called within MAGICSTAR. Refer to the control keys above to stop a recursive program.

What effect does the HOME command have on the program? Why was it used?

Practice Activities

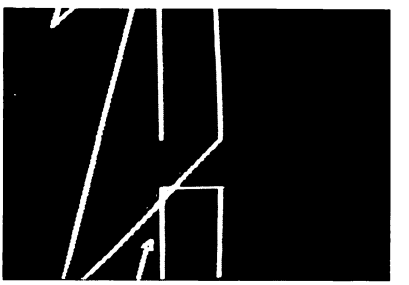
1. Write a program to draw a circle on the left of center screen and then erase itself. In the same program add lines to draw a circle on the right of center and erase itself. Continue this using a recursive command

2. Draw a vertical line from the top left of the screen to the bottom. Erase the line and draw one 20 steps to its right from the bottom to the top. Continue this pattern across the screen. Add some color to make it interesting.
3. Draw a small square which disappears and then appears elsewhere on the screen. **Hint:** Use PENUP and PENDOWN to manipulate the turtle without leaving a trail.

Setting Limits for the Screen

As you discovered earlier, the screen is made up of 240 vertical turtle steps and 280 horizontal turtle steps. Since the turtle's home position is in the middle of the screen, it is 120 steps from the top and the bottom. How far is it from each side? If you instruct the turtle to move forward 200 steps from home and it is 120 steps from the top of the screen, what do you think will happen? What happens is called *wrapping around.* Try these instructions and watch each carefully as it is executed:

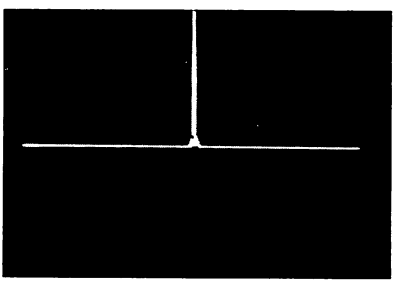
```
FULLSCREEN
FD 100
FD 100
RT 90
FD 50
RT 90
FD 200
RT 45
FD 100
FD 50
FD 50
RT 150
FD 300
```



You see that the turtle goes off the screen and comes back according to its position when it left. This is the *wrap*, just as though you were wrapping string around a piece of cardboard. The turtle automatically wraps around the screen unless you tell it differently.

Clear the screen and enter these lines:

```
WRAP
FD 121
BK 2
HOME
RT 90
FD 138
FD 2
HOME
```



Perhaps you have the following occurrence: the turtle is invisible at times when using the WRAP command. If this happens to you, it is because there is a buffer area surrounding the screen in which the turtle can hide briefly. By experimenting, you can determine just how large this buffer is. The turtle is automatically in the wrap mode when Logo is first loaded.

FENCE Command

FENCE instructs the turtle *not* to go outside of the screen boundaries—not to wrap. Give a command that would exceed the screen limits, such as FD 300:

```
CS
FENCE
FD 300
```

You will receive an error message that says

```
TURTLE OUT OF BOUNDS
```

The turtle will not even move. The screen will remain in the fence mode until you desire it to be in the wrap mode again at which time you would enter WRAP.

WINDOW Command

WINDOW allows the turtle to go off the screen but does not let it wrap. It continues in a straight path, like laying string across a piece of cardboard on a flat surface, except that there are limits to the length of the string. Clear the screen, enter WINDOW, and type these instructions.

```
FD 100
FD 100
```

The line did not wrap around, and you did not get an error message. The line is extended (theoretically) out from the screen.

Try the following instructions using the WRAP command. Clear the screen and enter WINDOW followed by the same instructions. Compare the difference in the execution of the instructions. Clear the screen and this time use FENCE. You can not get past the second line this time. Why? Because you will get an error message that the turtle is out of bounds.

```
FD 200
RT 90
FD 50
LT 90
BK 200
LT 90
FD 50
```

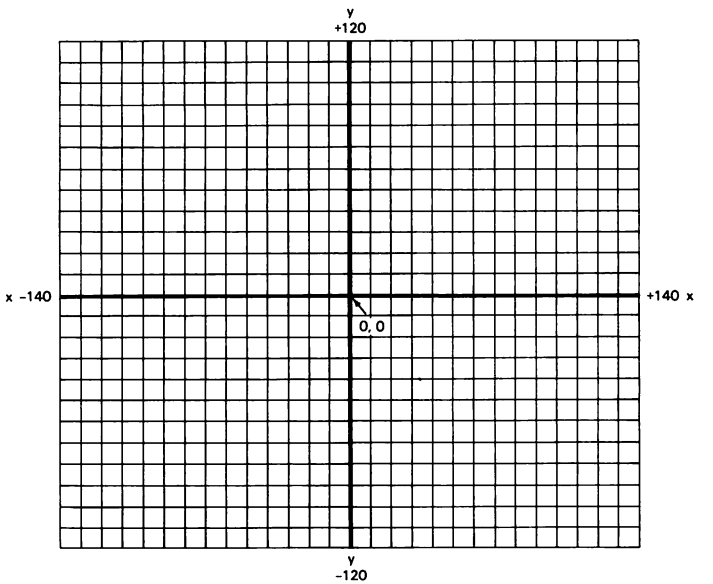
Practice Activities

Type WRAP before attempting the following activities:

1. Make a diagonal from top right to bottom left, long enough to wrap around. Predict where it will come back onto the screen.
2. Draw a circle in the center of the screen. Using a recursive, move it to the right 50 steps each time it repeats. Use WRAP, FENCE, and WINDOW in separate runs of this program.

Now We're Getting Somewhere

The turtle's home is in the center of the screen, at coordinates 0,0. Think of a vertical and horizontal line dividing the screen equally through the center. The first 0 refers to the horizontal or *x* axis, and the second 0 stands for the vertical or *y* axis. At this point it will be very helpful for you to get some graph paper to make sure you understand how to determine the coordinates to use in designs.



Earlier it was discovered that there are 240 vertical turtle steps from the top of the screen to the bottom. Horizontally there are 280 steps from the left edge to the right edge. On the grid above, each line represents ten turtle steps.

Practice Activities

1. If the center point is 0, then to get to the top of the screen what will the *y* coordinate be?
2. What would be the *x* coordinate for the far right edge of the screen?
3. What would be the *x* coordinate for the far left edge of the screen? (**Hint:** So far you have been using positive numbers.)
4. What is the *y* coordinate for the bottom center of the screen?

DOT Command

To verify your answers, we can use another command called DOT. The DOT command will place a dot anywhere on the screen, given two coordinates, *x* and *y*. It is in the following form:

```
DOT [0 0]
```

The command is followed by a space, and the *x* and *y* coordinates are inside the brackets separated by a space. Enter the line above, preceded by the command HIDETURTLE, and look for the dot in the center of the screen.

To place a dot in the upper right corner, the command would be as follows:

```
DOT [138 118]
```

The *x* or horizontal coordinate is on the right edge (+138) and the *y* coordinate is at the top (+118).

You might want to use the FULLSCREEN command so that the lower portion of the screen allows for full graphics. Use the DOT command to check your predictions above regarding the *x* and *y* coordinate screen limits. Remember that different monitors may treat the limits somewhat differently, and you must alter your use of coordinates accordingly. What if you wanted to refer to an area in the lower left quarter of the screen? Referring to the grid, anything towards the bottom is a negative number on the *y* axis, and anything towards the left edge is a negative number on the *x* axis. Therefore,

```
DOT [-138 -118]
```

puts the dot in the lower left corner of the screen. To place a dot in the lower right portion of the screen would require a positive *x* coordinate and a negative *y* coordinate:

```
DOT [75 -50]
```

Practice Activities

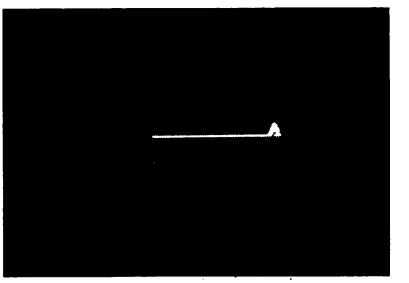
Place a dot in the following locations:

1. In the center top of the screen.
2. Halfway between the center and upper right corner.
3. Halfway between the center and lower left corner.
4. In the lower left corner.
5. In the upper right corner.
6. A terrific game to play is to place a spot on the screen using a felt pen and try to get as close to it using the DOT command as you can. Try it a few times to see how accurate you have become.

SETPOS Command

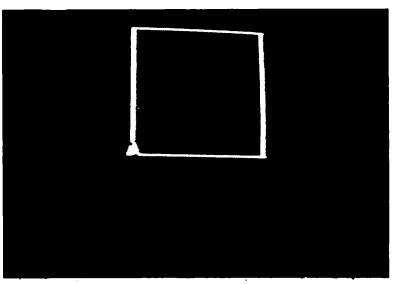
There is another command which manipulates the turtle using coordinates. SETPOS is its name, and it draws a line from the turtle's present location to the coordinates specified. Here is an example of its use:

```
CS
SETPOS [100 0]
```



When you type in these lines, the turtle actually draws a line from where it is to the coordinates specified. In this case, the line is drawn from the center of the screen, 100 steps to the right (positive direction) on the horizontal axis and 0 steps on the vertical axis. By entering the next three lines what will you end up with?

```
SETPOS [100 100]
SETPOS [0 100]
SETPOS [0 0]
```



Did you guess correctly? It is another way to make a square. In effect what the SETPOS command is doing is telling the turtle to draw a line from where it is to the new coordinates. Notice that when you give the turtle a command using SETPOS, the turtle does not face a new direction. It remains facing in its original direction, even if it is moving left, right, or down.

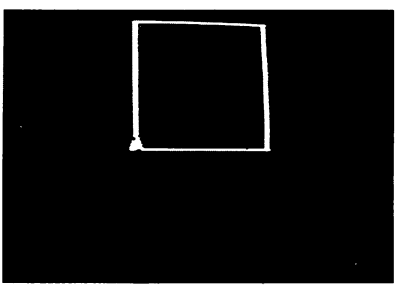
Practice Activities

1. Place four dots on the screen (don't make it too easy) and then use SETPOS to connect them. You will already have the coordinates identified so the SETPOS commands should be easy.
2. Make a six-pointed star using dots first and then filling in the lines.
3. Draw a square on the left part of the screen using SETPOS.
4. Draw a border around the entire screen. Make vertical and horizontal crosses through the center of the screen.

More with X and Y

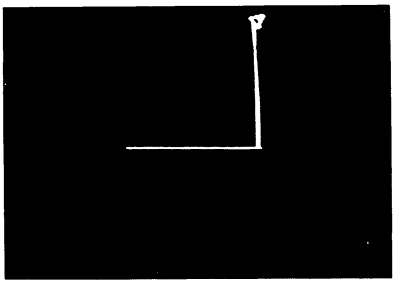
There is yet another way of producing a square (you are probably getting tired of squares, but you can compare this with other methods you have learned.) Look at this program:

```
SETX 100
SETY 100
SETX 0
SETY 0
```



Why does this one give us a square? Again, it is instructing the turtle to go to the spot representing +100 on the *x* axis (or horizontal line) and leave a trail. Next it tells the turtle to go to the location of +100 on the vertical axis from its current location. Refer to your graph paper if you are having any trouble visualizing this. The turtle just goes from where it is currently located to the place you specify with the SETX or SETY command. Using this system you can only get the turtle to move vertically or horizontally. For example, throw in a right turn and see what happens:

```
SETX 100
RT 45
SETY 100
```



The turtle changes its heading, but the line is still drawn from one coordinate to the next.

Practice Activities

1. Draw a horizontal line across the screen from left to right. Erase it and draw one above it. Include a recursive command so that this continues.
2. Do the same as in number 1 but make vertical lines.

SETHEADING Command

SETHEADING (or SETH for short) can also come in handy in all this coordinate business. SETHEADING makes use of specific or absolute values. For example, the command SETH 90 will cause the turtle to make a 90 degree turn to the right.

The command SETH 180 causes the turtle to point straight down.

If 0 were given as the heading it would point straight up. With right and left turns, the degrees always start from the turtle's current direction, but with SETHEADING, 0 is always straight up, 90 is always a right angle to the right, 180 is straight down, and 270 is a right angle to the left.

Enter the following commands and watch the turtle after each one to see the movement:

```
SETH 90
SETH 360
SETH 150
SETH 180
SETH 0
SETH 250
```

If you are unsure of what the turtle's heading is in a program, by using SETH you can place it at the heading you desire.

Practice Activity

Using SETH and FD draw another square. Can you put diagonal lines in the middle of it to connect the corners? Do it in the fewest number of steps possible.

SETHEADING TOWARDS Command

One last command is called SETH TOWARDS. It is a combination of the SETH and SETPOS commands. It tells the turtle to point in a certain direction using the coordinates x and y but does not cause it to move. The line

```
SETH TOWARDS [100 100]
```

will cause the turtle to face in a 45 degree angle to the right. Why? The command instructs the turtle to face the x coordinate, 100, plus the y coordinate, 100. The same would happen with

```
SETH TOWARDS [1 1]
```

Check your grid to see why this always produces a 45 degree angle.

Practice Activities

1. The program that follows is the beginning of a square. Enter these lines and then finish the square using the new commands:

   ```
   SETH TOWARDS [100 0]
   FD 100
   ```

2. Draw vertical and horizontal lines through the center of the square.
3. Make an outline of the screen. Then make a smaller box inside the outline and keep making it smaller. You might want to get fancy and make several procedures called OUTLINE1, OUTLINE2, and so forth, and then combine them into a procedure called OUTLINES and add some color.

Combining Procedures

8

With the amount of information that has been presented, programs can begin to be quite complex. Putting these concepts together, although not difficult, takes planning and thought. Examples will be shown that step you through the process of combining procedures, and several activities will be suggested for practice.

Combining Procedures

Programs that have already been introduced on pages 31 and 32 are needed for the activities in this chapter. If they have not been saved already, enter and save them at this time.

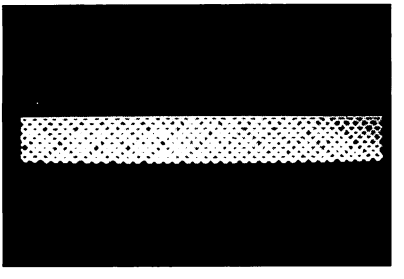
The problem we are going to work on is this: design a PLAYGROUND scene where there is a chainlink fence with a bouquet of balloons tied to it. Later, other items of your choice will be added to the scene.

CHAINLINK

The following program creates a graphic which resembles a fence. It will be called CHAINLINK. Why can it not be named FENCE? Any command or operation already defined in the Logo language cannot be used as a procedure name.

Although the entire procedure is listed below, it is suggested that you first enter the second through fifth lines in the turtle mode, observing each step as it is entered. This aids in the visualization of exactly what the program is doing. Then enter the procedure:

```
TO CHAINLINK
RT 90
FD 140
BK 280
REPEAT 37[FD 5 RT 45 FD 50 LT 90 FD 50 !RT 45]
END
```



Now let's examine the procedure line by line.

Line one: This is the title of the program.

Line two: Fences are parallel to the ground, so this 90 degree right turn prepares the turtle for the top of the fence.

Lines three and four: These draw the top of the fence by moving to the right of the screen and back to the left, where the turtle stops.

Line five:

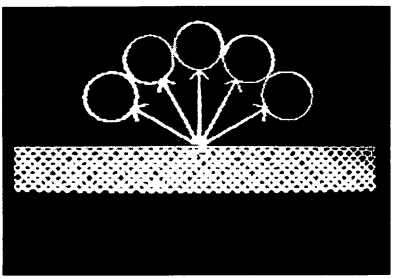
1. Immediately it is apparent that a set of instructions will be repeated 37 times. The number 37 was chosen using trial and error to determine what was necessary to complete the fence. The position of the turtle, before this line is executed, is at the far left of the screen, heading 90 degrees to the right.
2. The turtle travels 5 steps along the top rung and makes a 45 degree turn. It then proceeds forward 50 steps, makes a 90 degree left turn, and goes 50 more steps.
3. Notice the exclamation point after FD 50. It is placed there automatically by the computer when a line of instructions continues to the next line.
4. The last instruction in the REPEAT command is RT 45. This returns the turtle to the original heading of 90 degrees to the right.
5. The second REPEAT is now executed, beginning with the turtle moving 5 steps along the horizontal top rung. This is necessary each time so that the zigzag lines are not drawn on top of each other. The wrap feature is being used so that it is not necessary to insert a command to get the turtle back to the original position each time.

Line six: This is the END statement.

If you have not already done so, enter the procedure and run it. If all looks OK, it is time to coordinate it with the BALLOONS procedure. The goal is to simulate the balloons being tied to the fence. Because of the way the BALLOONS program was written, the turtle needs to be in the center of the screen, facing straight up, when BALLOONS begins. To accomplish this, one line needs to be added to the CHAINLINK procedure. Do you know what it will be? By entering HOME just before the END statement, the turtle is now ready for the BALLOONS procedure.

By combining the two procedures into a third, the entire scene can be drawn with the call of the composite procedure, which will be called PLAYGROUND. Be sure to type in the procedure named BALLOONS from Chapter 4.

```
TO PLAYGROUND
CHAINLINK
BALLOONS
END
```



The fence is drawn and the turtle returns to the center. The BALLOONS procedure is executed, and there you have it—your delightful bouquet of balloons is now tied to the chainlink fence. There are no limits to the kinds of scenes you can create using a combination of procedures as long as you are aware of their placement on the screen. Manipulation of angles is critical. Often the use of HEADING, POS, and *x* and *y* coordinates greatly facilitates the task.

Practice Activities

1. Make the fence white, the balloons blue, and the background black.
2. Add an orange kite to this scene in the upper left corner. Don't hesitate to use the PENUP, PENDOWN, and PENERASE commands if needed.
3. Draw a string from the kite to the ground under the fence.
4. Place an object on the ground for the kite to be tied to such as a seesaw, slide, tree, or post.
5. Add all these procedures to PLAYGROUND so that when you call for PLAYGROUND the entire scene is produced.

Refining Your Skills

The remainder of this chapter includes many different types of programs to write and problems to solve. Here is the opportunity to test and refine your skills. There are numerous ways to write each procedure, although some

programs are more efficient than others. By *efficiency* is meant the ability to write programs in the fewest number of lines possible. When writing a program, the goal is to write the program to accomplish its purpose as efficiently as possible.

Activity 1
ROOM

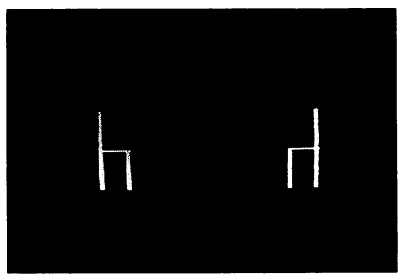
In this activity you will enter and edit three procedures: CHAIRS, TABLE, and PANE. Place them in a file named ROOM so that when ROOM is called to run, first the chairs appear, then the table between them, and finally the window in the upper right of the screen.

Note: If procedures are accidentally named using any word in the Logo language, a message will inform you of this fact:

```
(WHATEVER NAME YOU USED) IS A PRIMITIVE
```

If this occurs, a new name must be chosen. PANE is used above instead of WINDOW since WINDOW is a predefined operation. Following is the procedure for CHAIRS.

```
TO CHAIRS
HT
PU
HOME
FD 10
LT 90
FD 100
PD
LT 90
FD 60
BK 30
LT 90
FD 20
RT 90
FD 30
LT 90
PU
FD 125
LT 90
PD
FD 30
RT 90
FD 20
RT 90
FD 30
BK 60
END
```



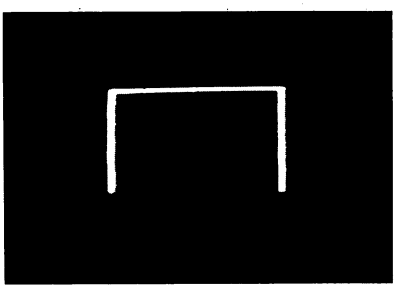
Make each chair a different color, with a third color for the background.

Practice Activity

The TABLE procedure is listed below.

Activity 2 TABLE

```
TO TABLE
PU
HOME
RT 180
FD 50
RT 90
FD 60
RT 90
PD
FD 50
RT 90
FD 75
RT 90
FD 50
RT 90
END
```



Practice Activities

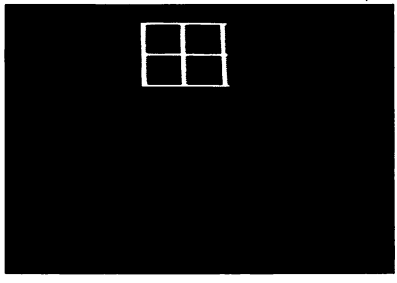
1. Alter the table so that it has a two dimensional look.
2. Use a pen color of your choice.
3. Run the CHAIRS and TABLE procedures to check their placement in relation to each other.

Activity 3 PANE

The third procedure to add to ROOM is a window which will be called PANE (since WINDOW is a primitive).

```
TO PANE
PU
HOME
FD 75
RT 90
PD
FD 50
```

```
LT 90
FD 40
LT 90
FD 50
LT 90
FD 40
RT 180
FD 20
RT 90
FD 50
BK 25
RT 90
FD 20
BK 40
RT 90
END
```



Practice Activities

1. Place the same window in the top left corner of the room. Do this without rewriting the entire procedure.
2. Make the window outlines a different color from the crosses through them.

Activities Using ROOM

1. Place these three procedures in a package named ROOM.
2. Save the package ROOM in a file also named ROOM.
3. Run the file named ROOM. Does it do what you expected?
4. Erase these procedures from the memory, but not from the disk, to prepare for the next problems.

Activity 4 SEASCAPE

Visualize a seashore, seagulls gathering above, and a couple of sailboats on the water. It does not take an artist to create lovely scenes. Seagulls can be large Vs, some wings opened wider than others. The seashore can merely be a diagonal line across the screen. The boat can be as simple

as a triangle or as complex as your imagination. Using graph paper can be helpful, but trial and error is sometimes quicker and often produces more interesting results.

Practice Activities

1. Design several birds at different locations, in various stages of flight. Color them white.
2. Draw a seashore, either a straight or jagged line, in blue across the screen. There are several ways to give the appearance of an irregular line. Can you discover one?
3. Place two boats on the ocean but use the same boat program for each. Color the boats green.

   Note: It is not possible to color portions of the screen with Apple Logo as is possible with some graphics languages such as PILOT.
4. After you are satisfied with the three procedures, place them in a package named SEASCAPE and save them in a file by the same name.

Activity 5 COUNTRYROAD

Here is another activity for you to try. The four components of this scene will be a ROAD, CABIN, MOUNTAINS, and TREES.

1. Make the road appear to taper off in the distance.
2. Place the cabin along the road, complete with chimney and little clouds of smoke coming from it.
3. Add a large mountain range in the background.
4. Make trees of two sizes. Place some at the base of the mountain and others along the road.
5. Choose an appropriate color scheme.
6. Place these procedures in a package named COUNTRYROAD and save it under that name.

9 Input and Variables

Input Variables

Two concepts that have not been introduced can add much versatility to your programs. They have extensive uses, only a few of which are presented in this chapter.

The first is the concept of a *variable*. If you plan to use a certain number in a procedure, you may give it a variable name and thereafter call it by that name whenever you want to use it. For example, you might assign the variable name SQUARESIDE to the number 25, or the variable name COLOR to the number 5. Whenever you want, you may change the name or number assigned to a variable. By using variables in Logo, you can increase the efficiency of your programs.

The second concept is *input*. This is what you have been doing every time you entered something into the computer. Inputs may also be used with variables in procedures to allow the user a certain amount of control over the running of the program. Let's look at some examples:

```
BK 50
```

Of course you recognize this line instructing the turtle to go backward 50 steps. The 50 is an input. Other commands, such as FD, RT, and LT also require inputs or they will not be executed. If you entered the following:

```
FD
```

you would receive the error message

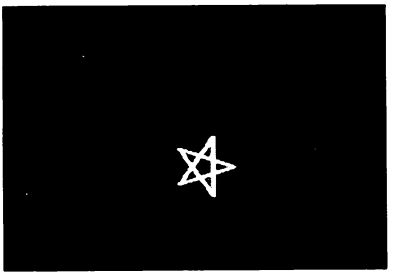
```
NOT ENOUGH INPUTS TO FORWARD
```

You can define your own commands, like FD, within procedures and require them to have inputs. Perhaps you would like to invent a command called STAR, so that when you enter

```
STAR 50
```

a star with sides 50 steps long will be produced. Here is what the procedure might look like:

```
TO STAR :SEGMENT
REPEAT 5[FD :SEGMENT RT 144]
END
```



Let's go through this one step at a time:

Line one:

1. TO STAR names the procedure followed by a space.
2. The colon signals the computer to expect a word as opposed to a number.
3. SEGMENT is the variable name given to the sides of the star. The variable named SEGMENT could have been SIDE or LINE or anything the programmer desires. Literally, line one is saying, "In the procedure named STAR, require the user to input a number after calling the procedure which will be the length of the sides of the star." Just like FD 50, STAR 50 will now produce a star with each side 50 steps long.

Line two:

1. REPEAT 5 is a familiar command: the computer will repeat what is inside the brackets five times. This will be a five-sided star.
2. FD :SEGMENT instructs the turtle to go forward the number of steps that was input. If the user entered STAR 50, the turtle would go forward 50 steps since the variable SEGMENT now has the numeric value of 50. STAR 75 would produce a star with longer sides.
3. RT 144 instructs the turtle to turn right 144 degrees.
4. Having finished the first execution of the instructions inside the brackets, it goes back for the second round.
5. The turtle goes forward 50 steps (SEGMENT always has the value of 50), turns right 144 degrees, and goes back for the third, fourth, and fifth times.

Line three: After completing the five repeats, the program ends. What would be the difference in the graphic pro-

duced if the user entered STAR 100? The star would be twice as large as STAR 50. STAR 25 would produce a small star.

What would happen if the user simply entered STAR? Since the procedure is written requiring one input, an error message would appear:

```
NOT ENOUGH INPUTS TO STAR
```

Whenever this message is received it is an indication that the procedure requires one or more inputs in order to be executed. Unfortunately, it does not tell you how many inputs are needed, but this can be discovered through experimentation. Procedure names can be chosen that tell the number of inputs. For example, STAR could be called STAR1 or STAR11 (for one input). This is completely up to the programmer.

Practice Activities

1. Write a program so that when the user enters LINE 78, a line is drawn from the center upwards 78 steps.
2. Answer the questions about the following procedure:

```
TO STAIRS :STEPSIZE
REPEAT 6[FD :STEPSIZE RT 90 FD :STEPSIZE LT 90]
END
```

   a. What is the name of the procedure?
   b. If the user enters STAIRS 20, where in the REPEAT command will the 20 be substituted?
   c. How many times will the 20 be used throughout the entire execution of the program?
   d. Draw an approximation of what this program will look like.

3. Write a program such that, when the line ZIGZAG 30 is entered, a line will zigzag across the screen from left to right, each line being 30 steps long.

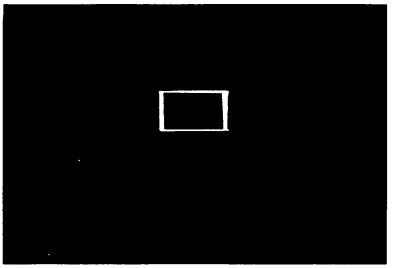
Multiple Inputs

It is completely acceptable to use more than one input in procedures. You might want to require values to be input for lengths of more than one line or for the number of degrees for a turn. Study this procedure:

```
TO RECTANGLE :LINE1 :LINE2
REPEAT 2[FD :LINE1 RT 90 FD :LINE2 RT 90]
END
```

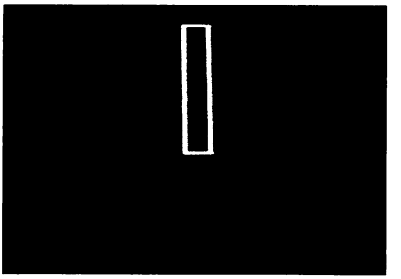
Here, the procedure requires two inputs, because after the procedure is named, there are two variables following it. The user must give two numbers when running it:

```
RECTANGLE 30 50
```



This will produce a rectangle that is wide and narrow.

```
RECTANGLE 100 20
```



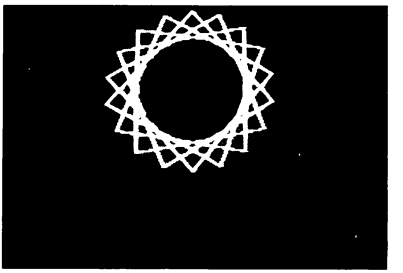
This will produce a rectangle that is tall and skinny. Do you see why? The turtle proceeds forward 100 steps, since LINE1 has the input of 100. It then turns right 90 degrees and goes forward 20 steps, the LINE2 INPUT. This is repeated to complete the rectangle.

What will this graphic look like?

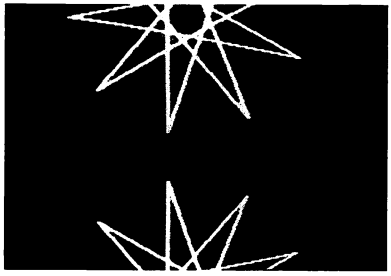
```
TO CRAZY :LINE :DEGREES
REPEAT 50[FD :LINE LT :DEGREES]
END
```

There is no way you can predict this program without knowing what the inputs are. Use these:

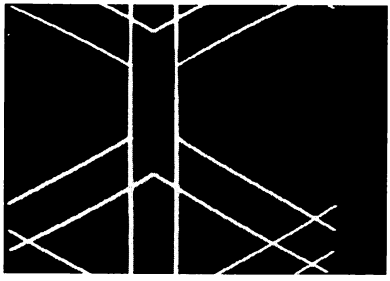
```
CRAZY 15 70
CRAZY 50 50
CRAZY 100 180
CRAZY 50 90
CRAZY 100 100
```



CRAZY 200 200



CRAZY 300 300



You know that the turtle will travel forward the first number of input and then turn left the number of degrees indicated by the second input number.

Practice Activities

1. Write a program to make a zigzag go across the screen to the left. This requires that the user give the length of the lines as well as the number of degrees the turtle turns.
2. Write a program requiring two inputs for stairs going from the bottom right to the top left of the screen. One input will be for the vertical line, the other for the horizontal line.
3. Write a procedure requiring three inputs: LINE1; RTURN (for right turn); and LINE2.

Procedures within Procedures Using Inputs

Just as with simple procedures, those requiring inputs can also be called inside each other. However, keep this in mind: the entire first line of a procedure is considered its name, so every time it is used, the whole line is necessary. Here are some examples:

```
TO STARS :SEGMENT
STAR :SEGMENT      (Make sure the STAR procedure is
                    in memory)
PU  LT 90
```

```
FD :SEGMENT
PD
STARS:SEGMENT
END
STAR 50          (Input)
```

Line one: Names the procedure and requires one input, called SEGMENT.

Line two: Calls the procedure named STAR (the same one used earlier in this chapter), along with its input variable, also called SEGMENT. If the input variable name associated with STAR were not the same as that for STARS, it would not be possible to combine the procedures in this way. Whatever is input when STARS is called will be substituted at each place SEGMENT is encountered. (To this point we now have one star on the screen.)

Line three: PU (PENUP) and LT 90 are found on the same line. This is perfectly legal and saves time. However, it gets confusing if too many commands are put on one line. The turtle ends up at a 90 degree angle to the left with the pen in the up position.

Line four: Causes the turtle to go forward the SEGMENT input amount. The pen is still up so this is just moving the turtle to a new position so that the stars are not drawn on top of each other.

Line five: Places the pen in the drawing position.

Line six: Calls for the procedure to start over—a recursive. Since SEGMENT is part of the procedure name, it must be included here. At this point, execution of the program starts over with line two, and draws a second star at the current location of the turtle.

Practice Activities

1. Write a program called BOX1 requiring one input to be used for each side. Include it in a procedure named BOXES1, which uses the same input variable name for the sides. Make the boxes appear at different positions on the screen and add some color for interest.
2. Write a program requiring two or more inputs. Include a recursive command.

Internally Generated Input

There is another way to generate numeric input besides requiring the user to do so. Before discussing it, some groundwork must be laid.

In Chapter 7 the concepts of SETH, SETPOS, SETX, and SETY were presented. There are four other operations

related to these commands. Their purpose is to inform the programmer about the location or heading of the turtle. The first of these operations is called HEADING, and it is the counterpart of the SETH (SETHEADING) command which, when followed by a number representing absolute degrees, will face the turtle toward that position. The HEADING command is used to determine the current direction the turtle is facing. Study these examples:

```
RT 90
PRINT HEADING
90.                    (Output on screen—notice decimal
                        point)
RT 180
PRINT HEADING
270.                   (Output on screen)
```

The second line utilizes two new words: PRINT and HEADING. The PRINT command is simply asking that the answer to the following inquiry (HEADING) be shown or printed on the screen. PR is the short form of PRINT.

HEADING, the inquiry, requests that the present heading of the turtle be identified. (Recall that a zero is straight up; straight down is 180, and so on.) After just making a 90 degree turn to the right in the first line, the HEADING output from line two is 90. The decimal point is always given. The output is on line three. The turtle then makes a 180 degree turn on line four so that it is facing directly to the left, and the heading is now given as 270., (or 90. + 180.). Heading is always given in degrees to the right.

Practice Activities

1. In the turtle mode, give the turtle at least three commands so that when you enter PRINT HEADING, the output will be 0.
2. Write a procedure so that the turtle turns 45 degrees and the HEADING is printed. Repeat this until the HEADING is 0.

POS

The POS operation is the counterpart of the SETPOS command. The SETPOS command, followed by x and y coordinates, directs the turtle to draw a line from its current position to the x and y coordinates indicated. The POS operation requests that the two coordinates be identified, similar to the HEADING operation.

```
CS
PR POS
0. 0.                    (Output on screen)
```

In this example, we have asked for the current coordinates of the turtle to be shown on the screen. Assuming the turtle is home, the output is 0. 0., which is centerscreen. Again, the values output on the screen are followed by decimal points.

```
SETPOS [100 50]
PR POS
100. 50.                 (Output on screen)
```

In these lines the turtle has been directed to travel to the x coordinate of 100 and the y coordinate of 50—thus the output 100. 50.

Practice Activities

1. In the turtle mode and without using SETX, SETY, or SETPOS, attempt to place the turtle at these sets of coordinates: 0 and 100; 100 and 0; 60 and −110; −50 and 50; −100 and −40. Use POS to check your accuracy.
2. Write a program which manipulates the turtle and periodically asks for its position to be output.

XCOR and YCOR

The two operations, XCOR and YCOR, are related to SETX and SETY. They report the x or y coordinates, respectively. For example,

```
HOME
PR XCOR
0.
PR YCOR
0.
```

Again, the center of the screen is 0. 0. The first coordinate is x and the second is always y. After moving the turtle around a bit, the x and y coordinates will be different:

```
FD 50
RT 90
FD 150
PR XCOR
-130.                    (Output on screen)
PR YCOR
50.                      (Output on screen)
```

Look at your grid (page 53) and follow the turtle's movement in the first three lines. The fourth line is asking for the x coordinate to be identified. It is shown on line five. The identification of the y coordinate is also requested and printed as output.

Practice Activities

1. In the turtle mode, move the turtle around the screen so that you receive the following outputs for the x coordinate: 20; -75; -115; 120.
2. Do the same thing you did in number 1 except these will be y coordinate outputs: 110; 55; -90; -10.
3. Write a procedure that makes use of the XCOR and YCOR operations.

Making Use of Internally Generated Input

When values for the heading, position, x or y coordinates are requested, the output can be used as input in programs. In order to change the output into input, a command called MAKE is necessary. To illustrate this, a series of examples are presented. Enter each line and watch what happens every time you press RETURN.

```
CS
LT 90
FD 100
RT 90
FD 50
MAKE "POSNOW POS
PR POS
-100. 50.                (Output)
PU
RT 90
FD 100
LT 45
BK 100 PD
PR POS
-70.7107 -20.7107        (Output)
SETPOS :POSNOW
PRINT POS
-100. 50.                (Output)
```

The first five lines are straightforward (except for the right turn). The sixth line, MAKE "POSNOW POS, can be literally described in this way: "Make a variable named POSNOW which takes on the value of the current position of the turtle." Thus in line seven, when PRINT POS is executed, the output is -100. 50., the location of the turtle after making the movements instructed in the first five lines.

In line nine the pen is raised and then the turtle takes off again. Note the output in line fifteen of the turtle's current position. Four decimal places are given.

Finally, line sixteen, SETPOS :POSNOW, can be literally translated: "Set the position of the turtle to whatever the position was when the variable POSNOW was made, which was -100. 50. (see line eight)." This is verified by the

PRINT statement in line eighteen, which responds with the coordinates -100. 50.

The MAKE command is a way to define a variable with any value you desire—even one which is determined as the program instructions are executed. The word MAKE must be followed by a space, quotation mark, the variable name, and the value you want the variable to take on. POS has a numeric value, so the variable POSNOW will contain the value in POS. Any variable name can be chosen, as long as it is not in the Logo vocabulary.

The current position is checked (line fifteen) after the changes to show that it will be in a different place at the end.

Another example makes use of heading information for the input: (Don't forget to put the pen down and clear the screen before entering these lines.)

```
LT 45
FD 100
RT 180
MAKE "HEADINGNOW HEADING
PR HEADING
135.                          (Output)
PU
FD 50
LT 90 PD
PR HEADING
45.                           (Output)
SETH :HEADINGNOW
PR HEADING
135.                          (Output)
```

The fourth line places the value for the current heading of the turtle into a variable named HEADINGNOW, saving it for later use. When the heading is printed, the absolute degrees is 135. The turtle is on the move again, but when it finishes it is facing 45 degrees. However, you instruct it to change its heading to what it was when HEADINGNOW was made in line four, which was 135. Immediately the turtle changes its heading to 135.

The same process can be followed using XCOR and YCOR. Go through these instructions: (**A reminder:** Put the pen down before entering this program.)

```
CS
FD 50
RT 90
FD 50
MAKE "XCORNOW XCOR
```

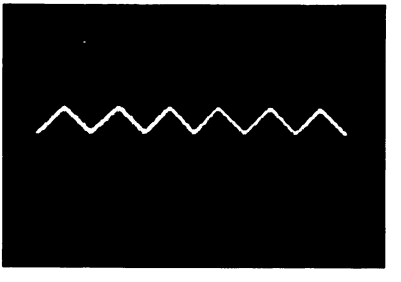
```
MAKE "YCORNOW YCOR
PR XCOR
50.                          (Output)
PR YCOR
50.                          (Output)
PU
RT 90
FD 50
LT 90
FD 50    PD
PR XCOR
100.                         (Output)
PR YCOR
0.                           (Output)
SETX :XCORNOW
SETY :YCORNOW
PR XCOR
50.                          (Output)
PR YCOR
50.                          (Output)
```

The same principle applies as has been described in previous sections, except that we are generating values from the XCOR and YCOR, which are placed in the variables named XCORNOW and YCORNOW. When the values are printed out, they are 50. 50. When the instructions SETX and SETY are encountered, they take on the value of whatever was placed in the variables XCORNOW and YCORNOW, returning the turtle to that position. The turtle actually backtracks with each command.

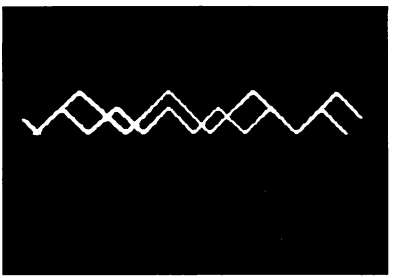
Using MAKE in a Procedure

Putting all this together, let's make use of the capabilities introduced above:

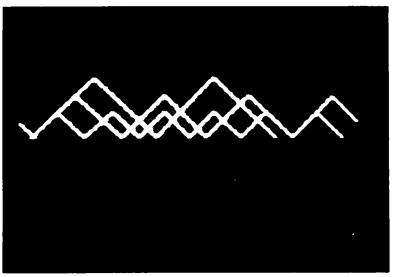
```
PU  LT 90  CS
FD 130
RT 90
MAKE "POSNOW POS
PD RT 45
MAKE "HEADINGNOW HEADING
REPEAT 6[FD 30 RT 90 FD 30 LT 90]
```



```
PU
SETPOS :POSNOW
SETH :HEADINGNOW
PD
REPEAT 4[FD 50 RT 90 FD 50 LT 90]
```



```
PU
SETPOS :POSNOW
SETH :HEADINGNOW
PD
REPEAT 2[FD 70 RT 90 FD 70 LT 90]
```



Each time the zigzag is drawn, the turtle returns to the position on the left of the screen before drawing the larger zigzag. It is also instructed to head in the appropriate direction, 45 degrees, which was set in the MAKE "HEADINGNOW HEADING command.

Practice Activities

1. Move the turtle to a position on the screen. Define that position using MAKE, move the turtle again, and return it to the first position (not centerscreen).
2. Move the turtle to a position and define the x and y coordinates using MAKE. Move the turtle's position again, returning it to the position where you placed it first.
3. Write a procedure to make a house similar to the one you made before. Define the starting position of the

turtle so that it returns there to finish the drawing. Also define the position at the base of the roof so that when you draw the horizontal line completing the roof triangle, it returns to the corner ready to draw the vertical side of the house.

Arithmetic Operations

10

Logo and Arithmetic

Built in to Logo is a calculator which allows arithmetic to be done. Not only can it add, subtract, multiply, and divide, but it can also find the square root, sine, cosine, randomize numbers, and more. These operations can be used with or without the turtle. Some examples of using the calculator without the turtle follow. Instructions to solve each problem are given in two forms, each of which will output the correct answer. Enter one form at a time and press RETURN.

Problem	*Instruction*
Add 5 + 10	PRINT 5+10 or PRINT SUM 5 10
Add 256 + 349	PRINT 256 + 349 or PR SUM 256 + 349
Subtract 6–3	PRINT 6–3 or PR 6 - 3
Subtract 199 - 78	PR 199 - 78 or PRINT 199-78
Multiply 5 × 5	PR 5 * 5 or PRINT PRODUCT 5 5
Multiply 99 × 13	PRINT PRODUCT 99 13 or PR 99 * 13
Divide 8 by 4	PRINT 8/4 or PRINT QUOTIENT 8 / 4
Divide 100 by 10	PR 100 / 10 or PR QUOTIENT 100 10

There is much flexibility in the format of the arithmetic statements:

1. Either PRINT or PR must be used for the answer to be given as output on the screen.
2. When adding, multiplying, or dividing, the operation can be indicated by the words SUM, PRODUCT, and QUOTIENT. There is no word for subtract.
3. When the words SUM, PRODUCT, and QUOTIENT are used, the respective signs (+, * and /) are optional.
4. Spacing is only important when the operation is called by name (as in QUOTIENT 9 3). In this case, a space must follow the operation word. Otherwise, it does not matter if there are spaces between the numbers and signs. (See table above.)
5. When adding or multiplying, if more than two numbers are involved, the sign (+ or *) must be used.

Practice Activities

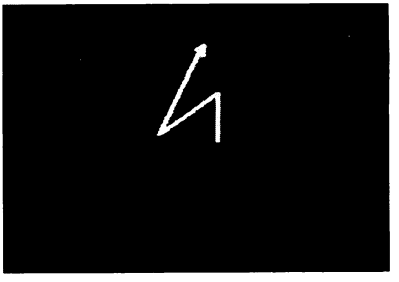
1. Find the sum of the following numbers: 334 + 555; 3456 + 4421; 4455 + 667 + 3456; 6.5 + 3.28; and 99.81 + 88.

2. Subtract these numbers: 76 - 34; 7684 - 4657; 36547 - 4357; 55.27 - 958; and 435.6 - 7.5.
3. Multiply: 5 × 6; 34 × 88; 32 × 66; 3 × 775; 71 × 225; 8.5 × 4432; and 14.2 × 6.4.
4. Divide: 100 ÷ 10; 78 ÷ 9; 345 ÷ 22; 5633 ÷ 56; .88 ÷ 3.3; and 55.672 ÷ .10.

Using Arithmetic Operations in Procedures

Integrating arithmetic operations within procedures is an added feature of Logo. In simple instructions, it can be used in this way:

```
FD 10 + 30
RT 100 - 45
BK 2 * 30
LT 90 / 3
FD 30 + 45
```



This is interesting but does not really accomplish anything. When used in procedures, however, all kinds of things can be done. In this simple program, an extra 10 steps is added to the initial number of steps input:

```
TO ADVANCE :STEPS
FD :STEPS + 10
END
ADVANCE 20                (Your input)
```

The turtle moves 30 steps since your input was 20 and 10 was added to it in the procedure.

What do you think would happen if we added a new third line:

```
TO ADVANCE :STEPS
FD :STEPS + 10
ADVANCE :STEPS
END
```

It is a little difficult to follow the turtle when the movement is so fast. Let's insert a WAIT command so that we can see exactly what it is doing:

```
TO ADVANCE :STEPS
FD :STEPS + 10
WAIT 120
ADVANCE :STEPS
END
```

The number with the WAIT command indicates how long you wish the program to pause before continuing. The number sixty is equal to one second. The turtle waits two seconds (WAIT 120). If your input is ADVANCE 20, how far does the turtle move the first time? Thirty steps. How far does it move the second time? It moves the same number each time. STEPS is always equal to 20. In the FD command, 10 is added to the input, which is always 20, so the turtle moves forward 30 steps.

Stop the procedure and edit it again:

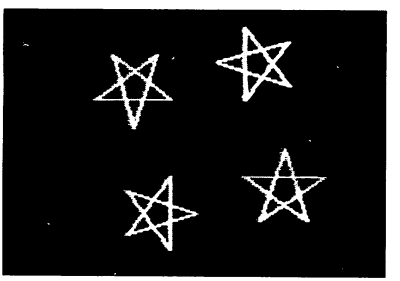
```
TO ADVANCE :STEPS
FD :STEPS
WAIT 120
ADVANCE :STEPS + 10
END
```

Enter ADVANCE 10 and press RETURN.

In this procedure, the actual value of STEPS is changed in the fourth line. Ten is added to the STEPS input value *each* time the loop is executed. Therefore, if the input was 10, the first time the turtle moves 10 (line two), the second time it moves 20, and so on.

Another way arithmetic operations can be used in procedures is shown in this program, which is an edited version of the STARS procedure on page 70 in Chapter 9. Be sure STAR on page 66 is loaded before running the following procedure, since STAR is called in it:

```
TO STARS :SEGMENT
PU LT 90
FD :SEGMENT / 2
PD STAR :SEGMENT
STARS :SEGMENT
END

STARS 40          (Input)
```



In the third line, the slash is used to indicate division. The only difference in this program and the old STARS program is that after one star is drawn, the distance between it and the next one will be half the distance indicated by the SEGMENT input. SEGMENT / 2 literally means to divide the segment input number by 2. Run this procedure with and without the / 2 to see the difference.

What would you change in this procedure to have the distance twice as far as the segment input? The third line would read:

```
FD :SEGMENT * 2
```

Run the program with this change.

Another example brings in a new trick. The pen color can be set using an input, too. In this program, the pen color is set to change each time. Enter and run it (you will get an error message but don't worry about it for now):

```
TO STARCOLORS :SEGMENT :COLOR
SETPC :COLOR + 1
FULLSCREEN
STAR :SEGMENT
PU  LT 90
FD :SEGMENT
PD
STARCOLORS :SEGMENT :COLOR + 1
END
STARCOLORS 25 1                          (Your input)
```

Why does the error message appear?

```
SETPC DOESN'T LIKE 6 AS INPUT IN STARCOLORS:
SETPC :COLOR  + 1
```

Since color numbers include 1 through 5, when the program has added enough ones to reach 6, the error message appears and the program stops. It also lists the line in the program where it stopped execution. This is helpful for debugging programs since it gives you an indication of where the problem is. There is a way to avoid this error message which will be introduced in the next chapter.

The program above is identical to the STARCOLORS program in the last chapter except that the pen color is determined by input as well as by the size of the star. The color value is changed each time by an increment of 1, thus making each star a different color. The first time it is run, the pen color is 2 (green), since 1 is added to the color value in the second line. The second time the color is 3 (green + 1), violet, and so on.

If the initial input was

```
STARCOLORS 25 4
```

what would happen? One star would be drawn in blue (4 + 1), and then the same error message would appear.

Practice Activities

1. Write a program to draw a rectangle. Have it drawn at a different angle each time by adding ten degrees to the input variable ANGLE.
2. Edit the STARS procedure so that each star is twice the size of the one before.
3. Edit the STARS procedure so that each star is half the size of the one before it and the distance between them decreases by 5 steps.

Random As an Arithmetic Operation

The capability of generating random numbers can be utilized in a variety of ways. The instruction is simply:

```
PRINT RANDOM 10
```

This means literally: "Print a number from 0 to 9 which is selected at random." Whatever number follows the RANDOM operation cannot be selected. Negative numbers *cannot* be generated. If you wanted a random number between 1 and 5, the line would look like this:

```
PRINT 1 + RANDOM 5
```

This means literally: "Print a random number from 0 to 4 but add 1 to it first." If the number generated was 0, adding 1 to it would make it 1. If it were a 4, the output would be 4 + 1, or 5. The number 5 is part of the possible numbers chosen since one is added. What if you wanted a number between 10 and 20 inclusive? The line would be

```
PRINT 10 + RANDOM 11
```

If the random number were 10, adding 10 would make it 20. If the random number were 0, adding 10 would make it 10. And those were our limits—from 10 to 20. Basically the formula for the selection of a random number within specific limits is

```
PRINT (lower limit) + RANDOM (upper limit + 1 minus lower limit)
```

According to this rough formula, how would you write the line if you wanted random numbers from 100 through 200? Between 350 and 375? The answers are as follows:

```
PRINT 100 + RANDOM 101
PRINT 350 + RANDOM 26
```

In the first line, the lower limit is 100. The upper limit + 1 (201) minus the lower limit (100) is 101. In the second example, the lower limit is 350; the upper limit + 1 (376) minus the lower limit (350) is 26.

Using Random Numbers in Procedures

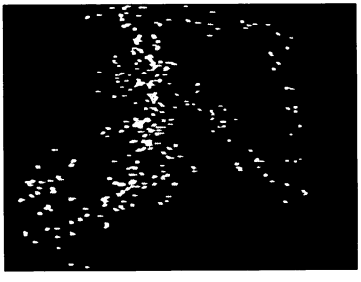
The STARCOLORS procedure can be altered once again to make use of the RANDOM operation. Compare this program with the one already written:

```
TO STARCOLORS :SEGMENT
SETPC RANDOM 6
FULLSCREEN
STAR :SEGMENT
PU LT 90
FD :SEGMENT
PD
STARCOLORS :SEGMENT
END
```

The main difference is that one input (color) is eliminated, and the pen color is determined by a randomly generated number from 0 to 5. Will you receive the same error message you did with the program in its unaltered form? No. Since the random number cannot exceed 5, the number 6 (which was the cause of the error message) will never be encountered.

Another way in which RANDOM can be used is with the commands FORWARD, BACK, RIGHT, and LEFT. This program will randomly place dots (equal to one turtle step each) on the screen:

```
TO DOTS
PU FULLSCREEN HT
LT RANDOM 360
BK RANDOM 25
PD
FD 1
DOTS
END
```



Let's look at this procedure line by line.

Line one: Names the procedure.

Line two: Calls for a FULLSCREEN, PENUP position, and the turtle is hidden.

Line three: A left turn from 0 to 359 degrees (determined randomly) is made.

Line four: The turtle moves backward from 0 to 24 steps (determined randomly). The pen is still up.

Line five: The pen is placed in drawing position.

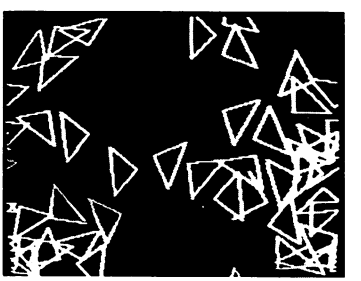
Line six: The turtle takes one step. The DOT command cannot be used here because the RANDOM operation cannot be used with it.

Line seven: The procedure is recursive so steps 1 through 6 are repeated.

Now run the procedure. There are a number of ways to write this same procedure, of course. What changes can be made that will give the same results?

Another example illustrates how shapes can be drawn randomly on the screen. The concepts are similar to the DOTS procedure:

```
TO PYRAMIDS
PU FULLSCREEN HT
RT RANDOM 360
FD RANDOM 100
PD
RT 45
FD 30
RT 90
FD 30
RT 135
FD 40
PYRAMIDS
END
```

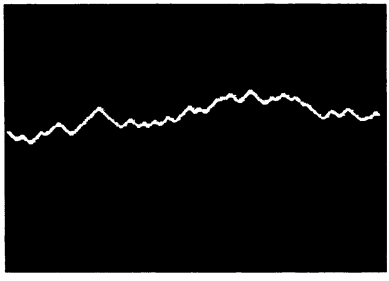


In lines three and four the position of the turtle is randomly moved. First, the direction (RT RANDOM 360) is changed, and then the location (FD RANDOM 100). Next the pyramid is drawn—lines seven through eleven. Then the procedure is repeated, with the turtle's position again being randomly changed. Run the program and observe its execution. Since it is recursive, you will have to stop it from running. Change the program so that each pyramid is a random color.

Getting a little fancier, enter these procedures:

```
TO PEAKS
REPEAT 50 [RT 90 FD RANDOM 15 LT 90 FD RANDOM 15]
END
```

```
TO HORIZON
FULLSCREEN FENCE HT PU
SETX -137
PD
RT 45
PEAKS
END
```



Practice Activities

The HORIZON procedure sets up a location for the turtle at the left center of the screen and turns it 45 degrees to the right before calling the PEAKS procedure. The PEAKS program then repeats the instructions in the brackets 50 times or until the edge of the screen is reached (FENCE). Inside the brackets the turtle is directed to zigzag across the screen, the length of each forward march being from 0 to 14 steps in length.

1. Edit the PEAKS procedure so that the forward moves are longer, simulating larger peaks.
2. Write a program to draw random-sized circles in random locations on the screen.
3. Write a program to draw squares, the length of the sides being dependent on an input, at random locations on the screen.
4. Add random color to the DOTS procedure.

Other Arithmetic Operations

A few arithmetic operations that are also available in Logo include the following:

- ARCTAN outputs the arctangent of the value input. The arctangent is the inverse of the tangent:

```
PR ARCTAN 55
88.9585          (Output)
```

- COS outputs the cosine of the value input (which refers to degrees):

```
PR COS 90
```

- INT (integer) deletes (or "truncates" in computer lingo) any numbers following the decimal point, outputting a whole number. It does not round the number off:

```
PR INT 34.5554
34
```

- REMAINDER indicates that two numbers must be input; the output is the remainder when the second is divided into the first:

```
PR REMAINDER 35 10
5                    (Output)
```

- ROUND rounds the number input to a whole number; five is used as the pivot number:

```
PR ROUND 76.8
77                   (Output)
PR ROUND 43.567
44                   (Output)
PR ROUND 543.432
543                  (Output)
```

- SIN outputs the sine of the input value, which is in degrees:

```
PR SIN 45
.707107              (Output)
```

- SQRT outputs the square root of the input value.

```
PR SQRT 81
9                    (Output)
PR SQRT 9
3                    (Output)
```

Practice Activities

1. Determine the arctangent of the following values; 35; 68.5; and 90.
2. What are the cosines of these values? 180; 90; 45; and 210.
3. Print the integer form of these values: 34.567; 20.1; 99.999; and 478.0002.
4. What are the remainders of these division problems? 34/6; 78/5; 673/25; and 4325/547.
5. Round the following numbers to the nearest whole number: 436.433; 84.5001; 35.499; and 99.99.

6. Print out the sines of these numbers: 36; 120; 99; and 150.
7. What is the square root of these numbers? 36; 360; 999; and 1001.

Conditionals, IF, AND, and OR

11

CONDITIONALS

In the last chapter an error message was received when the program STARCOLORS was run. The color number was incremented by one each time the procedure was repeated, and when color was finally equal to 6, the error message appeared:

```
SETPC DOESN'T LIKE 6 AS INPUT IN STARCOLORS:
```

A second way to eliminate that error (besides using RANDOM to limit the values for COLOR) is to request the computer to check the current value of a variable. When it reaches a certain value (which the user specifies), the user gives a different instruction. This capability is called a *conditional*, and the command IF initiates it. The procedure below is similar to STARCOLORS because it changes the color of the graphic each time the procedure is repeated. The IF instruction appears in line three:

```
TO COLORLINE :COLOR
FULLSCREEN
IF :COLOR = 6 [STOP]
SETPC :COLOR
FD 20
COLORLINE :COLOR + 1
END
COLORLINE 1
```

The third line literally means: "If the value (number) of COLOR is equal to 6, stop the procedure right here." The command STOP must be inside brackets. At this point, the screen will show a question mark, indicating it is ready for the user's next instruction. If the value for COLOR is *not* equal to 6, the computer ignores the command inside the brackets and continues on to the next line.

If the input were COLORLINE 2 how many lines would be drawn; COLORLINE 5; COLORLINE 6?

Enter these lines, clearing the screen before each one, and count the number of colors utilized:

```
COLORLINE 5
COLORLINE 2
COLORLINE 0
COLORLINE 6
```

When COLORLINE 5 is entered, only one line is drawn—blue. If COLORLINE 6 is entered no lines are drawn. How

does moving line three in the procedure below change the execution of the procedure?

```
TO COLORLINE2 :COLOR
SETPC :COLOR
FD 20
IF :COLOR = 6 [STOP]
COLORLINE2 :COLOR + 1
END
```

Can you predict the outcome using the same numeric inputs listed at the bottom of page 89? Will they be the same or different this time? The placement of the IF command is critical to a procedure, as you can see. In COLORLINE2, if the COLOR input is 6 to begin with, the error message that appeared previously will return. Do you see why? The command CHECKING to see if 6 was reached is not inserted until after the pen color is set, and that is what activates the error message.

A Different Use of Conditionals

Borrowing a concept from the COLORLINE procedure just described, a conditional can be used in a procedure designed to make the computer count. This concept can also be used to introduce two other operations. The following procedure causes the computer to output the numbers from 1 to 10, with the indicated user input:

```
TO COUNT10 :NUMBER
IF :NUMBER = 11 [STOP]
PR :NUMBER
COUNT10 :NUMBER + 1
END
COUNT10 1                    (User input)
1                            (Output)
2
3
4
5
6
7
8
9
10
```

The user initializes the value of NUMBER to 1 when calling the procedure. Immediately the conditional checks the value of NUMBER in line two: if the value of NUMBER input by user is equal to 11, then do what is indicated in the brackets—stop the execution of the procedure. If the value is not equal to 11, continue with the next lines.

The third line instructs that the number be output on the screen, and the fourth line increases the value of NUMBER by one. At this point, the procedure is repeated, beginning with the IF command, which again checks if the input number is equal to the value of 11. Since at this time the value of NUMBER is 2 (the initial 1 plus the 1 added in line four), 2 is output on the screen and one more is added to the value of NUMBER (making NUMBER equal to 3). This is repeated until the value becomes 10, it is printed on the screen, and one is added to it. When it is checked in the IF command, the computer realizes that it now fulfills the condition of being equal to 11 and does what is in the brackets—stops. How would the output change if the conditional line were placed after the PRINT operation? How could you change the IF operation so that you can place it at line three but still have the same results as in the initial COUNT procedure? Merely change the line to read

```
IF :NUMBER = 10 [STOP]
```

The number is output before the conditional is reached, so by changing the value of the number you are checking it against, anything over 10 will not be output.

What if the user input is

```
COUNT10 5
```

What will the output be? The screen will show the numbers

```
5
6
7
8
9
10
```

Whatever number is input will be the first number output. Going back to the original COUNT procedure, line two can be replaced by this line without altering the output:

```
IF 10 < :NUMBER [STOP]
```

Literally this line means: "If the value 10 is less than the input number then stop." If the input number is 1, the computer will check to see if 10 is less than that number (which it is not), so it goes to line three and prints 5. One is added to the value, and each time it is checked to see if 10 is less than the value. When the value is finally 10, the question is, "If 10 is less than 10 then stop." But 10 is *not* less than 10, so the number 10 is printed. The next time, however, the value will be 11, and "10 is less than 11," so the procedure is stopped.

Follow the same logic when the same line is written in this way:

```
IF :NUMBER > 10 [STOP]
```

This is very similar to the first conditional, since being greater than 10 is the same as being equal to 11. They both are true at the same time and will terminate the procedure at the same time.

There is one exception to this statement. When the user input is COUNT10 20, what is the difference in the execution of the procedures when these IF commands are inserted, one at a time?

```
IF :NUMBER = 11 [STOP]
```
or
```
IF :NUMBER > 10 [STOP]
```

There is a big difference! In the first line, if the number input is already greater than 11, the program will never be stopped since the number is always increasing. However, if the second line is included, any number input that is over 10 will stop the procedure.

The purpose in showing the different ways of writing the same line is to show the interchangeability of the variable name and value. The variable name can come first.

```
IF :NUMBER = 10 [STOP]
```

or second

```
IF 10 = :NUMBER [STOP]
```

The colon must be included immediately before the first letter of the variable name to alert the computer to expect a name as opposed to a number. This is true whenever words or characters are input as variables.

Practice Activities

1. Write a procedure to count from user's input to 20. Edit the IF command, putting it in a different sequence to produce the same results.
2. Write a procedure to count backwards from user's input and stop at 0. Write the IF command in three different ways.

IF with OR

In the examples introduced above the value of the variable was checked against just one other value. It is possible to check it against more than one value. If the goal of the procedure is to count to 100, only if the input value is greater than 49, how would this conditional be written? The IF line might look like this:

```
IF OR :NUMBER > 100 :NUMBER < 50 [STOP]
```

Literally, this line means: "If the number input is greater than 100 or less than 50, then stop." If just one of the conditions specified along with :NUMBER is met, then the program will do what is specified in the brackets—stop. The OR refers to "one *or* the other" condition must be true, or the procedure continues with the next line. If both are true, the instruction in the brackets is still executed. Of course, in the example given both cannot be true. However, there are occasions when two or three conditions could be true.

What happens if the line above is inserted into this procedure:

```
TO COUNT100 :NUMBER
IF OR :NUMBER > 100 :NUMBER < 50 [STOP]
PR :NUMBER
COUNT100 :NUMBER + 1
END
```

What is the output with the following user inputs?

```
COUNT100 50
COUNT100 100
COUNT100 30
COUNT100 200
```

In the first line, neither condition is true, so the output will be the numbers 50 to 100. In the second line, again neither condition is true since 100 is not greater than 100. The output will be 100. In the third line, the second condition is true so the program stops. The last line also meets one of the conditions—the first one where the input is greater than 100—and the program stops.

Practice Activities

1. Edit the COUNT100 procedure so that it only counts between 100 and 200.
2. Write a procedure that counts from 1 to 100 beginning with the user's input, and adding five each time the procedure is repeated. For example, if the input is 78, the output would be

   ```
   78
   83
   88
   93
   98
   ```
3. Write a procedure to count backwards from 50 to 10 beginning with the user's input.
4. Write a procedure that counts backwards from 200 to 150, subtracting two each time the program is repeated.

IF and AND

Use of the AND operation requires that *both* of the conditions be true as specified in the IF command. With line two in the following procedure *both* operations would never be true, no matter the value of the input number.

```
TO COUNT :NUMBER
IF AND :NUMBER > 100 :NUMBER < 50 [STOP]
PR :NUMBER
COUNT :NUMBER + 1
END
```

No number can be both greater than 100 and less than 50, so the instruction in the brackets would never be executed. What about substituting this line for line two:

```
IF AND :NUMBER > 49 :NUMBER < 100 [STOP]
```

Can both conditions be true? What would be the output if the input is

```
COUNT 101
COUNT 39
COUNT 99
COUNT 75
```

This procedure only counts if the input number is *not* between 50 and 100. In other words, if the number is between 50 and 100, thus meeting both conditions, the instruction in the brackets is executed. Any number input not between 50 and 100 will initiate the count procedure which will continue until you stop it or until it reaches 49.

Practice Activities

1. Edit the COUNT procedure so that it does not count if the input number is between 100 and 200.
2. Write a procedure that does not count if the input number is between 10 and 20. If it is over 19, it stops counting at 50.

Animation

12

Simulating Movement

As mentioned later in Part III, on some computers there are *sprites*, objects that are easily animated. Apple Logo does not have sprites at this time; the capability is available on the Krell version of Logo. However, there are some crude ways to simulate animation without sprites.

The Moving Truck

One way to simulate animation is to draw the object in one position on the screen and then erase it and immediately draw it again a short distance away from where the first one was. In other words, you might create a design of a truck and place it in a procedure. A second procedure instructs the truck to be drawn, erased, and then drawn again a few steps forward. This is repeated several times. Here is how it might look:

```
TO TRUCK
HT FD 15
RT 90
FD 20
RT 90
FD 5
LT 90
FD 5
RT 90
FD 10
RT 90
FD 25
RT 90
END
```

This is a simple drawing of a truck without wheels (they take too long to draw). To make it move across the screen, enter this procedure:

```
TO MOVETRUCK
CS
REPEAT 100[TRUCK PU CLEAN RT 90 FD 4 LT 90 PD]
END
```

In the REPEAT command, the truck is drawn, screen cleaned, pen lifted, and the turtle is turned 90 degrees before moving 4 steps forward. The 90 degree turn is necessary since the turtle ends up facing straight up, and we want the truck to move to the right, not up. After the pen is moved

to the right, it turns left 90 degrees to face up for another truck drawing and the pen is placed on the screen. This whole procedure is repeated 100 times, an arbitrary number.

Adding an angle to the procedure should make it more interesting. If the turtle is positioned in the upper left of the screen, it can move to the lower right, looking like it is going downhill. Edit the MOVETRUCK procedure to include four new lines:

```
TO MOVETRUCK
CS PU
SETPOS [-140 60]
RT 20
REPEAT 100 [TRUCK PU CLEAN RT 90 FD 4 LT 90 PD]
END
```

Run the procedure to see the difference.

Practice Activities

1. Edit the MOVETRUCK procedure so that the truck is going uphill.
2. Design an object and make it move upward.

An Underwater Scene

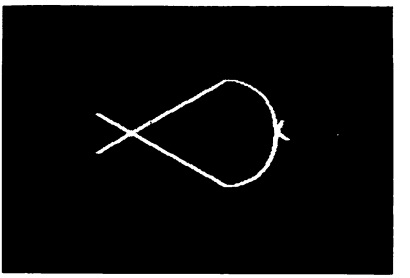
A second way to simulate animation is to continue drawing an object a few steps apart without erasing the one before. This is appropriate for certain designs. Here is an example:

```
TO BUBBLE
HT FULLSCREEN
ARCR 2 360
END

TO BUBBLES
REPEAT 3[BUBBLE PU RT 45 FD 15 PD BUBBLE PU
    LT 45 FD 15 PD]
END
```

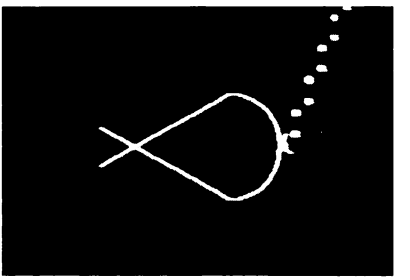
When running the BUBBLES procedure, it appears as though the bubbles are rising. Someone might look at it and think, "Why not put a fish, or bathtub under the bubbles?" This is the creative process which Logo automatically sparks.

```
TO FISH
ARCL 40 90
LT 30
FD 110
PU HOME PD
RT 180
ARCR 40 90
```

```
RT 30
FD 110
PU HOME PD
RT 30
FD 10
PU HOME PD
RT 120
FD 10
PU HOME PD
END
```



```
TO UNDERWATER
FISH
BUBBLES
END
```

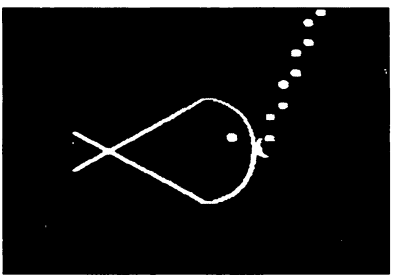


Perhaps the fish would look a little better with an eye. That shouldn't be too hard:

```
TO EYE
PU
SETPOS [-20 10]
PD
ARCR 2 360
PU HOME PD
END
```

Now we need to add the EYE procedure to the UNDERWATER procedure:

```
TO UNDERWATER
FISH
EYE
BUBBLES
END
```



The procedure named UNDERWATER is made up of three other procedures. First the fish is drawn, then the eye and bubbles.

Practice Activities

1. Edit UNDERWATER to add a water line above the fish. Put a plant of some type near the bottom.
2. Add a small fish to UNDERWATER with tiny bubbles (maybe just dots).

The Hungry Worm

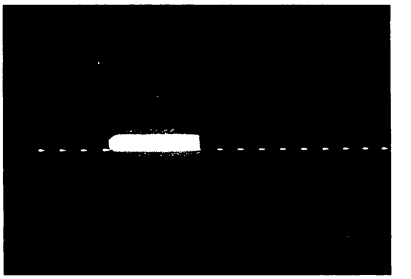
This example of animation uses the erasing technique but does not erase the entire object, just the rear portion, while adding to the front portion. This gives the impression of a slowly creeping worm. Add some worm food and we have a happening! Five procedures are used for this simulation:

```
TO FOOD
REPEAT 18[PU FD 15 PD FD 1]
END

TO WORM
REPEAT 5[RT 90 FD 1 LT 90 FD 61 RT 90 FD 1 RT 90 FD 61
    RT 180]
RT 90
BK 10
LT 90
END

TO HEAD
FD 60
RT 30
REPEAT 2[FD 5 RT 60]
FD 5
RT 30
FD 60
RT 90
FD 10
RT 90
END

TO MOVE
HEAD
BK 1
RT 90
PE
FD 10
BK 10
```

```
LT 90
FD 1
PD
FD 1
REPEAT 150[MOVE]
END
```

The last procedure, HUNGRYWORM, puts it all together.

```
TO HUNGRYWORM
HT WRAP CS
PU BK 5
PD LT 90
FOOD
WORM
MOVE
END
```



Practice Activities

1. Edit the HUNGRYWORM procedure so that the food is different—tiny plants or foliage lying on the ground.
2. Add some background to the HUNGRYWORM scene—a sun or other items.
3. Create your own animation. Here are some suggestions: a plant growing; paw marks going across the screen; rain or snow falling; or the screen filling up with water.

13 Text Use

Logo's main claim to fame is its graphics capabilities, and, as has already been shown, Apple Logo has arithmetic and conditional capabilities associated with turtle graphics. It also encompasses its own vocabulary for manipulating characters and words, which can be combined with graphics if desired. The text vocabulary is introduced briefly in this section. Limitations of this Primer do not permit the authors to do justice to the text manipulation capabilities of Logo. For further information refer to the manuals supplied with your version of Logo.

Some Words about Words

Characters, numbers, or letters that make up a *word* are collectively called *elements*. There are four elements in the word *draw*, three elements in the word *$50*, and five elements in the word *12ABC*.

In addition, there are special characters which cannot be used in the words unless the computer is alerted. These special characters are ones you will recognize:

( ) < > + - = * / [ ]

They have been assigned their own definitions in the Logo language. Therefore, in order to use them as ordinary symbols in words a control character must be used to alert the computer. This is done by typing a slash in front of the special character—but not an ordinary slash. The CTRL Q produces the special slash:

Correct Word	*How the Computer Reads and Prints It*
THANKSGIVING	*THANKSGIVING*
ONE\-WAY	ONE-WAY
(SECRET)	Error—I DON'T KNOW HOW TO SECRET
\(SECRET\)	(SECRET)
BJ+WN	Error—no slash before the plus
BJ\+WN	BJ+WN

To the computer, as soon as a space is encountered, the word is considered complete. If more than one word is going to be entered, they must be enclosed in brackets, and then they become a *list*. Each word in a list is also an element.

One Word	*List*
APPLELOGO	[APPLE LOGO] (Two words)
12345	[12 34 5] (Three words)
ABC123	[ABC 123] (Two words)
HFC46PM	[HFC 46 PM] (Three words)
TURTLEGRAPHICS	[TURTLEGRAPHICS] (A one word list)

PRINT with Text

The computer can read words or lists in different ways. Just entering

```
THANKSGIVING
```

will produce the error message

```
I DON'T KNOW HOW TO THANKSGIVING
```

The user must signal the computer to expect a word or list. The brackets are the signal for a list. For a word, there are three signals:

"WORD

When the word itself is the input, the word must be preceded by quotation marks.

```
PRINT "EXTRAORDINARY
EXTRAORDINARY          (Output)
```

The word EXTRAORDINARY is the word we wish to have printed, therefore it is the input information. Any word (no spaces allowed) preceded by quotation marks is treated as input and printed as is:

Input	*Output*
PRINT "1234	1234
PR "WATERFALL	WATERFALL
PR "OVER EASY	Error—no spaces are allowed—it is now a list of two words and must be enclosed in brackets
PR "EASY\-DOES\-IT	EASY-DOES-IT
PR "POS	POS
PR POS	Outputs a number indicating turtle's position
PR "$14.00A	$14.00A
PR "THIS "IS "EASY	I DON'T KNOW WHAT TO DO WITH IS (Only one word can appear with quotes)

More than one word cannot be entered using quotation marks, or the error message shown in the last example is output.

[WORD]

One word in brackets signals a one-word list. Even though there is just one word identified, PRINT followed by a word inside brackets works in the same way as with a quotation mark:

Input	*Output*
PR [9999]	9999
PR [OOPSIDAISY]	OOPSIDAISY
PR [*GREAT*]	*GREAT*
PR [YOU ARE CUTE]	YOU ARE CUTE (A list of words)

:WORD

A word preceded by a colon indicates that the word is the name of a variable. Therefore, this line

```
PRINT :STEP
```

will output the error message

```
STEP HAS NO VALUE
```

unless it is used within a procedure and the user has entered a value for STEP. If STEP does already have a value, the output would be that value, not the variable name STEP.

```
TO MOVE :STEP
FD :STEP
PRINT :STEP
END

MOVE 25      (User input—turtle moves 25 steps forward)
25           (Output)
?
```

Another Use for Brackets

In the procedure below, lines three, five, and seven cause an empty line to be left between the list (inside the brackets) being output:

```
TO SONG :NAME
PR [HAPPY BIRTHDAY TO YOU]
PR []
PR [HAPPY BIRTHDAY TO YOU]
PR []
PR [HAPPY BIRTHDAY DEAR] PR : NAME
PR []
PR [HAPPY BIRTHDAY TO YOU!]
END
SONG "CLEMENTINE      (User input)
```

The computer output is:

```
HAPPY BIRTHDAY TO YOU
HAPPY BIRTHDAY TO YOU
HAPPY BIRTHDAY DEAR
CLEMENTINE
HAPPY BIRTHDAY TO YOU!
```

Manipulating the Text

The PRINT command can be combined with certain operations that will cause the words and lists entered to be altered. For example, the operation BUTLAST (BL for short) will output *all but the last* element of a word or list. For example, in the line PRINT BUTLAST "LAST the output is LAS. Here are further examples:

Input	*Output*
PRINT BUTLAST "54321	5432
PR BL [ONE TWO THREE]	ONE TWO
PR BL [ONLY]	No output since ONLY is the last element
PR BL [[PLEASE PASS] [THE MILK]]	PLEASE PASS

The opposite operation of BUTLAST is called LAST, where *only the last* element is printed:

Input	*Output*
PRINT LAST "54321	1
PR LAST [ONE TWO THREE]	THREE
PR LAST [ONLY]	ONLY
PR LAST [[PLEASE PASS] [THE MILK]]	THE MILK

Closely related to BUTLAST and LAST are BUTFIRST and FIRST. They manipulate the words and elements at the beginning of the word or list. The abbreviation for BUTFIRST is BF. Its output is *all but the first* element:

Input	*Output*
PRINT BUTFIRST "54321	4321
PR BUTFIRST [ONE TWO THREE]	TWO THREE
PR BF [ONLY]	No output since ONLY is the first element
PR BF [[PLEASE PASS] [THE MILK]]	(THE MILK)

The FIRST operation outputs only the first element or word:

Input	*Output*
PR FIRST "54321	5
PR FIRST [ONE TWO THREE]	ONE
PR FIRST [ONLY]	ONLY
PR FIRST [[PLEASE PASS][THE MILK]]	PLEASE PASS

Counting the Elements

There is an operation called COUNT which can be used to determine the number of elements that make up a list. If you are unsure how many elements you have, enter the inquiry in this way:

```
PR COUNT [[PLEASE][PASS][THE][MILK]]
4                                    (Output)
PR COUNT [12345]
1                                    (Output)
PR COUNT [I[AM]FINE]
3                                    (Output)
```

WORD and SENTENCE Operations

The operation WORD combines the elements or words into one word, eliminating the spaces. Only words and *not* lists can be manipulated with this operation. If more than two words are to be combined, a set of parentheses must be around the operation *and* words:

Input	*Output*
PRINT WORD "EVERY "ONE	EVERYONE
PR (WORD "BOOK "KEEP "ER)	BOOKKEEPER
PR (WORD "A "O "K)	AOK
PR WORD ["UP "STAIRS]	Error message—WORD DOESN'T LIKE [UP STAIRS] AS INPUT
PR WORD "UP "STAIRS	UPSTAIRS
PR (WORD "UP "STAIRS)	UPSTAIRS

A similar operation, called SENTENCE (SE for short), handles words by putting them into lists. Then it puts the words or lists together to make a SENTENCE. The spaces between the words remain. Study these examples:

Input	*Output*
PRINT SENTENCE [BOOK KEEP ER]	BOOK KEEP ER
PR SE [SPRING HAS SPRUNG]	SPRING HAS SPRUNG
PR SE [[SPRING][HAS][SPRUNG]]	[SPRING][HAS][SPRUNG]

READCHAR and READLIST Operations

Two additional operations in the Logo language are similar to two BASIC commands called GET and INPUT. READCHAR (or RC for short) waits for the user to enter one character or number. As soon as any key is pressed, the procedure continues. In the following program, as soon as the user enters a number between 1 and 10 a special message is produced:

```
TO CHOOSE
PR [PICK A NUMBER BETWEEN 1 AND 10]
MAKE "NUM READCHAR
IF :NUM > 5 [PR[THAT'S A LARGE NUMBER]STOP]
PR [YOU MUST LIKE SMALL NUMBERS]
END
```

If a letter is entered instead of a number, an error message will appear. If more than one number is entered, the computer will act on the first number entered as soon as the key is pressed.

The READLIST operation waits until the user presses the RETURN key before executing the next instruction. RL is the short form for READLIST. In this program, the user's name is used in the output:

```
TO GREET
PR [WHAT IS YOUR NAME? ENTER IT AND PRESS
    RETURN]
MAKE "NAME READLIST
PR [HELLO,] PR :NAME
END
```

Line three could have been MAKE "NAME RL without any difference in the output. When using READLIST it is usually a good idea to instruct the user to press the RETURN key or else he may be sitting there a long time.

Practice Activities

1. Write a program with the user's first, middle, and last names, but which prints out only the user's initials.
2. In a procedure, ask the user to enter a number from 1 to 6. If it is over 3, make one comment; and if it is under 3, make another.
3. Write a program that utilizes the BUTLAST operation to remove the S from plural words, leaving the singular form.
4. Using the BUTFIRST and LAST operations, write programs to show your understanding of the use of those operations.
5. Using the WORD and SENTENCE operations, write programs to show your understanding of the use of those operations.

14 Combining Text and Graphics

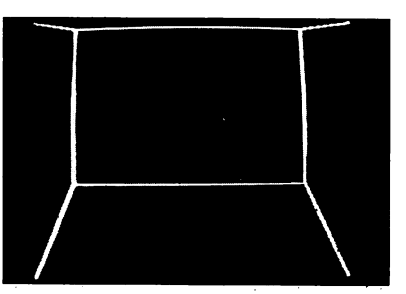
PROGRAMS USING TEXT AND GRAPHICS

Games and instructional programs can be programmed using Logo. The beauty is that graphics can be included in the programs with relative ease as compared to other languages. The use of text is a little more awkward but can be done. This chapter presents a shortened version of a game of perception and memory. You will see how easily it can be changed, expanded, or used as a model for other programs.

The master procedure includes three graphic representations of rooms with four text procedures integrated within. All of the procedures will be listed. Enter them all before running them since they are interrelated. A discussion will follow.

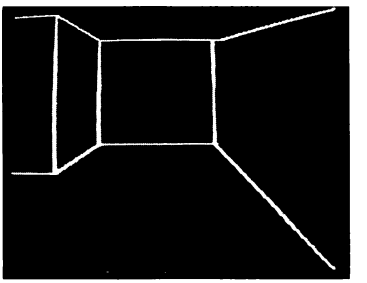
Large Room

```
TO ROOM1
HT CS PU FULLSCREEN
SETPOS [-140 119] PD
SETPOS [-104 110]
SETY -30
SETPOS [-140 -120] PU
SETPOS [-104 -30] PD
SETX 104
SETPOS [140 -119] PU
SETPOS [104 -30] PD
SETY 100
SETPOS [140 119] PU
SETPOS [104 110] PD
SETX -104 PD
WAIT 400 PD ANSWER1
END
```



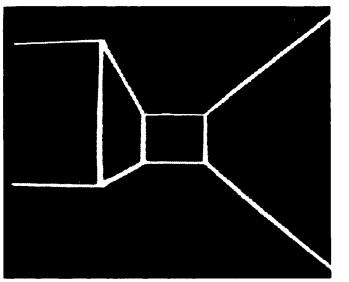
Medium Room

```
TO ROOM2
HT CS PU FULLSCREEN
SETPOS [-140 110] PD
SETX -104
SETPOS [-68 80]
SETY -5
SETPOS [-104 -30]
SETX -140
SETX -104
SETY 110 PU
SETPOS [-68 -5] PD
SETY 80
SETX 40
SETY -5
SETX -68 PU
SETPOS [40 80] PD
SETPOS [139 119] PU
SETPOS [40 -5] PD
SETPOS [139 -119]
WAIT 400 PD ANSWER2
END
```



Small Room

```
TO ROOM3
CS PU HT FULLSCREEN
SETPOS [-140 85] PD
SETX -68
SETY -40
SETX -140 PU
SETPOS [-68 -40] PD
SETPOS [-32 -20]
SETY 20
SETPOS [-68 85] PU
SETPOS [-32 20] PD
SETPOS [32 20]
SETPOS [32 -20]
SETPOS [-32 -20] PU
SETPOS [139 119] PD
SETPOS [32 20] PU
SETPOS [139 -119] PD
SETPOS [32 -20]
WAIT 400 PD ANSWER3
END
```



The Game

This procedure initiates the game:

```
TO ROOMGAME
CS HT
PR []
PR [I WILL SHOW YOU A ROOM AND THEN]
PR []
PR [ASK YOU SOME QUESTIONS ABOUT IT.]
WAIT 400
ROOM1
END
```

First Question

This procedure asks the first question and provides for the answer:

```
TO ANSWER1
SPLITSCREEN PR [] PR [] CS
PR [HOW MANY LINES DID THE ROOM HAVE?]
MAKE "ANS1 READCHAR
PR [] PR []
IF :ANS1 = 8 [PR "CORRECTO!] [PR [IT HAD 8 LINES.]]
PR []
PR [TRY THIS ONE...]
WAIT 400
ROOM2
END
```

Second Question

This procedure asks the second question and provides for the answer:

```
TO ANSWER2
SPLITSCREEN CS
PR [WAS THIS ROOM BIGGER THAN THE FIRST?]
PR [Y OR N?]
MAKE "ANS2 READCHAR
PR []
IF :ANS2 = "N [PR [YOU ARE RIGHT!]] [PR[SORRY BUT IT
WAS SMALLER.]]
PR [HERE IS ONE MORE...]
WAIT 400
ROOM3
END
```

Third Question

This procedure asks the third question and finishes the game:

```
TO ANSWER3
SPLITSCREEN CS
PR []
PR [ARE YOU FAR FROM THE WALL?]
PR [Y OR N?]
```

```
MAKE "ANS3 READCHAR
PR [] PR []
IF :ANS3 = "Y [PR "IT CERTAINLY IS.] [PR[YOU ARE
PRETTY FAR!]]
WAIT 400
PR [] PR []
PR [THAT WAS AN EXAMPLE OF A LOGO GAME.]
PR [I HOPE YOU ENJOYED IT.]
END
```

Discussion

Some important things to take note of in these procedures include:

SPLITSCREEN: It is necessary to include this when text is used after fullscreen graphics, or the text will not show up on the screen.

WAIT: This command should be tested with the users to determine an acceptable time period. It is also possible, using an additional instruction of "press RETURN when ready to continue," to eliminate the WAIT command. This gives more control to the user.

PR []: Spaces are inserted strategically so that previous text does not appear on the screen as the game progresses.

Logo makes heavy use of procedures within procedures. It is not possible to leave one procedure in the middle, go to another procedure, and return where it left off in the first procedure. A third procedure must be written. However, since they are easy to write and less confusing than long, branching programs, they are especially simple to combine.

Practice Activities

1. Using procedures named SQUARE, TRIANGLE, and DIAMOND, design a tutorial to teach the difference between them. Require occasional input of answers by the user to check understanding.
2. Write an adventure game set in a forest, with different consequences depending on the direction the player chooses to travel.
3. Remember the suggested game in the previous chapter in which the user places a dot on the screen with a felt pen and then tries to place a turtle dot at that location using coordinates? Write the game into a master procedure, allowing for the user to play until wishing to stop.

QUICK-START LOGO

This section of the manual is intended for the use of those who have had previous experience with programming languages, other versions of Logo, or strong egos. Logo is presented in a brief, concise manner. It should not be expected that it is exhaustive of Logo commands, operations, and capabilities. However, it offers the user a crash course which can be comprehended in a relatively thorough and quick manner.

Concepts presented in Part II are explained at much greater length in Part I, and it is suggested that the user refer to appropriate sections if questions or misunderstandings arise. The inside covers of this manual, appendixes, and glossary/index are also handy reference sources.

Getting Started

To begin using Logo follow these steps:

1. Insert the Logo disk into the disk drive.
2. Turn on the Apple microcomputer—the power switch is on the back left of the micro.
3. Turn on the monitor.
4. Press the RETURN key. Ignore the message regarding inserting your own disk. Wait until the screen is blank except for the message.

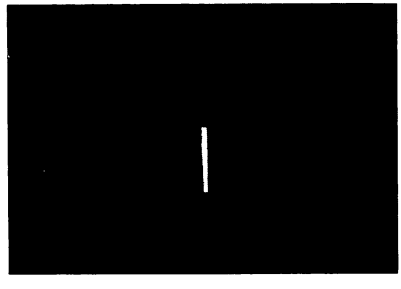
   ```
   WELCOME TO LOGO
   ?
   ```

   A white flashing box appears in the upper left corner next to the question mark. It is called the *cursor*.
5. If you make typing errors, or typos, just type the line over. If you discover an error before pressing RETURN, back up the cursor using the left arrow (find it on the right side of the keyboard) and retype the line from the point of error.
6. Type in the word SHOWTURTLE (or ST for short) and press RETURN. You are now ready to create some turtle graphics. What in the world are turtle graphics? I'm glad you asked.

Turtle Steps

In the center of the screen is a turtle (you may have thought it was just a triangle). The turtle knows how to move about the screen going forward and backward. The command FORWARD (or FD for short), followed by a space and a number, tells exactly how many "steps" you want the turtle to take in the direction in which it is pointing. Now it is pointing straight up. Type in this command (and always remember to press the RETURN key after you are finished typing a line):

```
FD 50          or          FORWARD 50
```



The turtle just marched 50 turtle steps forward. Make it go 25 more steps forward. Try having it go just one step and watch carefully as you press RETURN to see if you can determine any movement. One turtle step is very small.

The turtle can go backward too. FORWARD and BACK are the two most important commands in Logo. The command BACK (or BK for short) causes the turtle to move the opposite direction from which it is heading. Enter this line:

```
BK 155          or          BACK 155
```

Notice that the turtle went backward past the point it began by 79 steps. What happens if you instruct it to go back 200 steps? Were you right? It moves so quickly you can hardly tell that it is moving down the screen and coming back on the top—since it ran out of space at the bottom of the screen. This is called *wrapping* around the screen.

Try to get the turtle into the center of the screen again. Just backtrack the instructions you have already given. An easier way to do this is to type the word CLEARSCREEN (CS for short), which is quicker than backtracking since it clears the screen and places the turtle in the center of it.

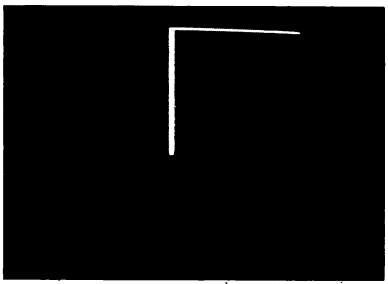
This clever turtle can also make turns to the LEFT (LT for short) and RIGHT (RT for short). Whatever direction the turtle is pointing, it will turn the number of degrees you tell it, in the direction you indicate (to the left or right). Enter these lines and observe the turtle as you press RETURN each time:

```
CS
RT 90
LT 90
RT 180
LT 360
```

The last line causes the turtle to turn completely around. The movement is so fast that it can hardly be perceived.

Combining these four commands, FORWARD, BACK, RIGHT, and LEFT, the Logo user can design an unlimited number of graphic objects. If you wanted to draw a square, here are some lines that you might begin with:

```
CS
FD 100
RT 90
FD 100
RT 90
```



Finish this program by adding lines that instruct the turtle to complete the square that is begun. Once you have mastered the square, experiment with other objects. (Don't forget to use the CS command each time you begin a new design.)

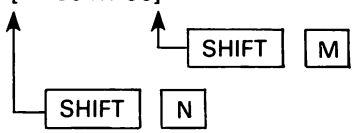
Practice Activities

1. How many turtle steps is it from the top of the screen to the bottom of the screen?
2. How many steps is it from the left side to the right side of the screen?
3. How many turtle steps is a diagonal across the screen?
4. About how many turtle steps does it take to make four inches?
5. How long is a turtle step?
6. Draw a rectangle—long and skinny; short and fat; outlining the screen (don't worry about lines appearing in the middle).
7. Draw a triangle—large; small; all sides equal.
8. Draw the letter N; T; H; make your initials.

Repeating Yourself

Check out this command:

```
CS
REPEAT 4[FD 50 RT 90]
```

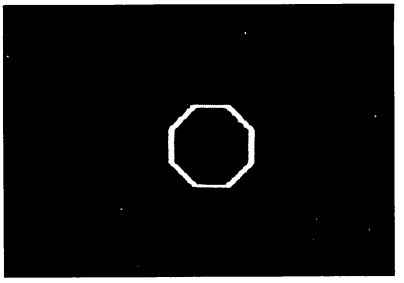


The square brackets in this line are special Logo characters made by holding down the SHIFT key and pressing the N for the left bracket and the M for the right bracket. Parentheses cannot be substituted.

If you want to draw a square, the above program is one way to do it. It saves a lot of time compared to entering a

series of FORWARD and RIGHT moves. The REPEAT 4 command tells the number of times you want the turtle to do whatever is inside the brackets. First the turtle goes forward 50 steps and then it turns 90 degrees; the second time around the turtle follows the instructions inside the brackets again, going another 50 steps forward and turning 90 degrees; and it does the same two more times and stops because it only was supposed to do it 4 times. What happens if you change the 4 to 8 and also change the numbers within the brackets? Enter this

```
CS
REPEAT 8[FD 25 RT 45]
```



You can make some interesting designs using the REPEAT command.

Practice Activities

1. Draw a large circle. **Hint:** A circle is merely a step followed by a turn followed by a step, and so forth.
2. Draw a small circle.

Fooling Around with the Turtle

You already learned SHOWTURTLE (ST is the abbreviated form). It is used to cause the turtle to appear when it is not on the screen. There is also the opposite command called HIDETURTLE (or HT for short). See what happens when you enter this command and press RETURN. The turtle is still in the same place, but you just can't see it. How can you prove this last statement? Simply type a FORWARD or BACKWARD instruction, indicate a number of steps, and watch what happens. Now make the turtle reappear.

Another fun thing you can do with the turtle is to tell it when to draw and when not to draw. PENUP and PENDOWN are turtle commands that stay in effect until you give the next pen command. When you first load Logo, the pen is down, ready to draw. When the pen is up, the turtle will move, but no drawing will occur.

Refer to the following table:

Abbreviated Instructions	*Nonabbreviated Instructions*
CS	CLEARSCREEN
ST	SHOWTURTLE
PU	PENUP
RT 90	RIGHT 90
FD 75	FORWARD 75
PD	PENDOWN
BK 75	BACK 75
LT 100	LEFT 100

On the left is the short version and on the right is the long version. Both programs will produce the same thing. (Use the abbreviations whenever possible. There are quick references to abbreviations on the inside covers.)

PENERASE

PENERASE is another pen command which can also come in handy.

Enter one of these four line programs:

CS	CLEARSCREEN
FD 80	FORWARD 80
PE	PENERASE
BK 80	BACK 80

If you draw a line you don't like, simply erase it by going back over it using the PENERASE command. Draw a square and then erase it. (It's not fair to use CLEARSCREEN!)

PENREVERSE

One additional pen command is PENREVERSE. The short form is simply PX. This command is tricky. It erases any line it goes over and draws on anything that is blank. In other words, it is actually reversing anything on the screen that it comes into contact with. As you enter each line below, watch the screen carefully:

```
PX
FD 80
BK 80      (Reverses line two since the turtle has left a trail)
FD 80
RT 90
FD 60
LT 90
FD 15
LT 135
FD 150     (Erases the points on the lines already drawn
            where it crosses them—otherwise it draws on the
            blank screen)
```

After using the PX command you must indicate your desire to change this pen mode by entering PD or PU or the PENREVERSE stays in effect. At this time type PD to eliminate the PX mode.

Practice Activities

1. Draw a shape and then erase it using PENERASE; do the same using PENREVERSE.
2. Move the turtle to a spot on the screen without leaving a trail and draw a shape. Return the turtle to the center (don't use CS).
3. Create a house beginning with a rectangle and placing a triangle on its top.

Keys That Control the Cursor

The cursor is an important symbol on the computer because it is always there to tell you your location. There are some helpful control keys that, when pressed after you hold down the key labeled CTRL located on the right side of the keyboard, produce special movements of the cursor. If you make a typing error and discover it before you press RETURN, you can move back or forward over the line to make the correction.

- CTRL B moves the cursor back one step (not the turtle).
- CTRL F moves the cursor forward one step.
- CTRL D deletes the character under the cursor.

For example, clear the screen and enter this line as you see it and press RETURN.

```
FF 100
```

The error message appears

```
I DON'T KNOW HOW TO FF
```

This is the way in which you are informed that you have made an error. (The error messages in Logo are considered to be quite polite compared to other languages!) Enter FF 100 again, but *don't* press RETURN. To correct the error, hold down the CTRL key and press the B. Each time you press the B, the cursor moves back one space over each character. Move it all the way back to the second F and *not* holding down the CTRL key, type a D. The line now reads

```
FDF 100
```

with the cursor on the second F. By holding the CTRL key

and pressing D, the F will be deleted. You have now corrected the line. Now use CTRL F until the cursor is over the 1 and change it to a 2. Delete the 1 (CTRL D) and press RETURN. The instruction is now executed. Experiment with both the CTRL B and F.

Keys and Words That Control the Screen

There are also control commands that manipulate the screen. Remember that each time you type a control key, you must first hold down the key marked CTRL.

CTRL L (FULLSCREEN)

CTRL L (FULLSCREEN) allows the entire screen to be used for graphics. For example

```
CS
BK 100
CTRL L      (Watch the screen as you press RETURN)
```

The turtle moves downward until its trail disappears due to the four lines at the bottom alotted to text. The CTRL L is the screen command that allows the use of the entire screen space for graphics, and you can see now where the line goes.

CTRL S (SPLITSCREEN)

CTRL S (SPLITSCREEN) is the screen mode you have been using. In this mode, graphics are on the top lines with four lines for text at the bottom. Type

```
CTRL S
```

and the screen again shows four lines of text at the bottom, cutting off the turtle trail that was drawn.

CTRL T (TEXTSCREEN)

CTRL T (TEXTSCREEN) clears the entire screen for text use only. Enter the following:

```
CS
CTRL T      (Hold down the key marked CTRL and type a T)
```

Immediately the screen is filled with instructions that have been entered. As soon as one turtle instruction is entered, the screen reverts back to a splitscreen. Now enter

```
RT 180
FD 100
CTRL L      (Fullscreen—pause here and notice the change)
CTRL S      (Bottom four lines are back on the screen)
```

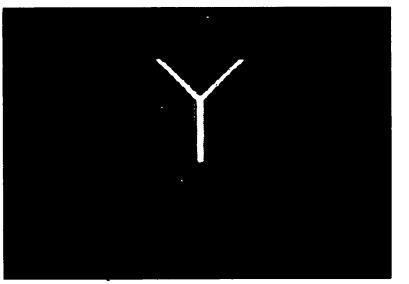
Enter the three screen control commands in their unabbreviated form: FULLSCREEN, TEXTSCREEN, and SPLITSCREEN. You will get the same results as when the CTRL key is used along with the appropriate letter: L, T, or S.

Note: When in the fullscreen mode, if you make a typo the screen reverts to a splitscreen in order to give the error message.

HOME

An additional screen command that comes in handy is HOME. This command places the turtle back in the center, but it differs from CLEARSCREEN: it does not erase anything but leaves a trail. This might be used when you are writing a program that moves the turtle around the screen, and then you want to start a new design from the center point again without tampering with what is currently on the screen. Since it does leave a trail, if one is not desired use the PENUP command before HOME. Try this program:

```
CS
FD 45
RT 45
FD 45
PU
HOME
PD
FD 45
LT 45
FD 45
PU
HOME      (The turtle is now back in the center)
```



Practice Activities

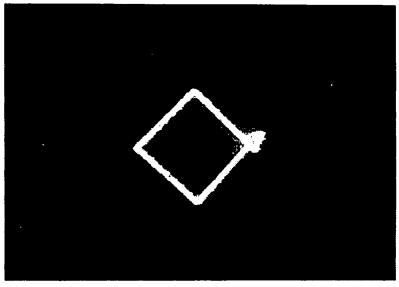
1. Draw different size triangles by drawing only two of the lines and using the HOME command.
2. Use PENUP and PENDOWN commands to place the turtle in each corner and then use HOME to draw a line from them to the center.

Keys That Control the Turtle

If you ever want the turtle to stop what it is doing, use CTRL G (This gives the message, "Stop, turtle!"). This comes in handy, especially when you use the REPEAT command. Enter these lines:

CTRL G

```
CS
LT 45
REPEAT 100[FD 50 LT 90]
```



The turtle is beating a mad path around the diamond. To stop this program from running, hold down the CTRL key and type a G. The following message appears:

STOPPED!

CTRL W

If you want to make the program pause, you should use CTRL W. When you are ready for the program to continue, simply type any character on the keyboard. Enter the above program again and use the CTRL W to make it wait and type any key to make it carry on. Stop the program using CTRL G and clear the screen.

Adding a Splash of Color

Basically, the turtle can draw in six colors if a color monitor is being used. (If a black and white monitor is being used, the background will be black with white lines. Skip to the next section.)

Think of the turtle as carrying a pen. To change the color of the lines or objects being drawn, the command SETPC (for set pen color) is used. Using the program above, add this as the first line:

```
SETPC 5     (This sets the pen color at number 5)
```

The program is drawn in color number 5, which is blue. There are six pen colors in Apple Logo:

0 = black	3 = violet
1 = white	4 = orange
2 = green	5 = blue

Change the pen color in the above program to check out each different shade available. If a color doesn't look like you think it should, you might want to adjust the color controls on the monitor. One caution—you must specify the pen color before any drawing is done.

You can also use these colors for the background or screen. Black is the automatic screen background color.

Clear the screen and type in these lines:

```
SETBG 1
SETBG 2
SETBG 3
SETBG 4
SETBG 5
```

The following program uses both pen color and background color:

```
CS
SETBG 4
SETPC 1
RT 45
FD 80
SETBG 1
SETPC 5
FD 40
```

Try the other colors in this program until you find pen colors and background colors that go well together. Some colors just don't mix, and the quality may depend on your monitor. Use different combinations of pen colors and background colors.

```
CS
SETPC 1     (White on white)
FD 70
LT 100
HOME
SETBG 5
BK 100
SETBG 2
SETPC 1
RT 90
FD 100      (Produces a white line)
SETBG 3     (Reverses again)
SETPC 4
FD 100      (Double line)
SETBG 4
LT 90
SETPC 5
FD 100
```

The Black and White Monitor

As with color monitors, the screen is automatically black with a white pen. Although the magic of color is not possible, there are some interesting effects that can be achieved on a black and white monitor. Enter these lines:

```
CS
FD 50
RT 90
FD 75
HOME
```

Watch what happens when you enter this line:

```
SETBG 1
```

This reverses the screen and pen colors. Experiment with this concept using your own designs.

Placing Programs into Memory

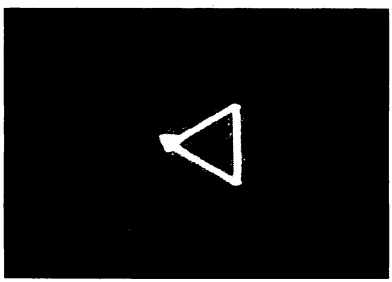
So far you have written short programs and run them immediately. There is a way to put your programs into memory; that is, a way to write the program so that after it is run once it is there to run again and again.

Note: Unless you just loaded Logo, enter

```
SETBG 0
SETPC 1
```

As you enter the following lines, the graphics will not be on the screen. In the procedure mode, only text lines appear. Because of this, you should have already perfected the graphic program before you write a procedure:

```
TO TRIANGLE
RT 30
FD 50
RT 120
FD 50
RT 120
FD 50
END
```



The keyword in this program is TO, which indicates that you want the procedure name and the instruction lines placed into memory. Enter the lines above; press RETURN; and type TRIANGLE and the procedure will be produced.

Note: If you make an error or there is a bug in a procedure, it will not be indicated until you run it. For example, in a procedure named TRI, if you have a line that reads FF 10, the following message will appear when you run it:

```
I DON'T KNOW HOW TO FF IN TRI:
```

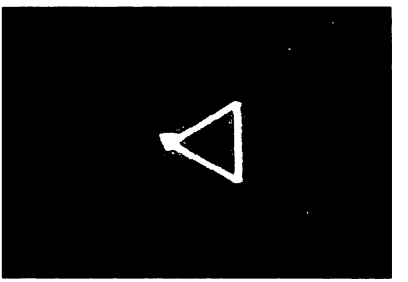
Soon you will learn how to make alterations in your procedures, called *editing*, to allow for additions, deletions, or corrections. (See page 125.)

After you enter END the screen will say TRIANGLE DEFINED. END signals the completion of the procedure and is a necessary part of the process.

Type TRIANGLE over and over, and the turtle keeps drawing triangles. Why aren't they all in the same place? It is due to the direction which the turtle is heading at the end of the program.

You may not use the name TRIANGLE for any other procedure or you will be in big trouble. Check the section on saving programs on page 127 in this chapter for clarification of the naming process. Here is the entire program, including your input and the screen output:

```
TO TRIANGLE          (Names the procedure)
RT 60                (The program instructions follow)
FD 50
RT 120
FD 50
RT 120
FD 50
END
TRIANGLE DEFINED     (Message on screen)
?
TRIANGLE             (Type this in to run the program)
```



The triangle appears! The screen does not automatically clear without the explicit instructions, CS. It is a good idea to include it in procedures if appropriate.

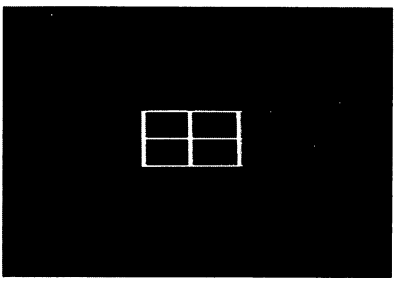
Practice Activities

1. Write a procedure to draw a table.
2. Write a procedure to draw a chair.
3. Write a procedure to draw a window (but call it PANE).

Procedures within Procedures

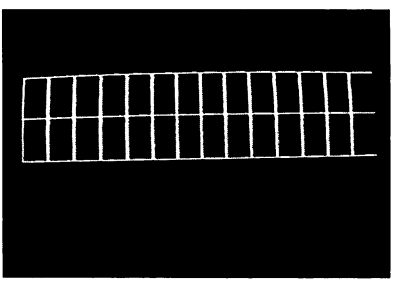
Here is a more sophisticated use of procedures:

```
TO PRESENT
FD 40
RT 90
FD 70
RT 90
FD 40
RT 90
FD 70
BK 35
RT 90
FD 40
BK 20
RT 90
FD 35
BK 70
END
```



Type in the procedure name PRESENT to see the graphic design before entering the following procedure:

```
TO PRESENTS
PU              (Lifts the pen)
LT 90
FD 100
PD              (Lowers the pen)
PRESENT         (Runs the PRESENT procedure above)
PRESENTS        (Sends the turtle back to the beginning of
                 PRESENTS starting with the PENUP
                 command)
END
```



The first procedure PRESENT produces a picture of a present on the screen. The second procedure, named PRESENTS, has the turtle travel somewhere on the screen,

draw a present, and then go somewhere else and draw another present, and on and on. The practice of including the procedure within itself is called *recursive*, which means that it will go on and on. It is like an infinite loop, if you are familiar with BASIC.

Use CTRL G to stop the procedure, and the screen output is

```
STOPPED! IN PRESENT:
RT 90      (Or whatever line it was executing when CTRL G
            was entered)
```

Practice Activities

1. Create a design in a procedure using rectangles and draw them repeatedly. (**Hint:** REPEATedly.)
2. Make a procedure that includes the subprocedures of chair, table, and pane. Place them appropriately on the screen.
3. Write a procedure that is recursive. Don't forget how to stop the turtle. (**Hint:** CTRL G.)

Editing Your Procedures

Everyone has a right to change their mind, and the turtle agrees. Therefore, if you want to change a procedure in any way after you have written it, here is what to do:

1. Type ED "PRESENT or EDIT "PRESENT (ED is short for EDIT). It must be followed by a space, a quotation mark, and the name of the procedure.
2. Your program instructions will now be shown on the screen. At the bottom of the screen it will say LOGO EDITOR. Use the following control characters to manipulate the cursor by holding down the key marked CTRL and the letter specified.

CTRL A	Cursor moves to *beginning* of line
CTRL B	Cursor moves one space *back*
CTRL C	Puts your *changes* in memory and gives the message "PRESENT DEFINED"
CTRL D	*Deletes* the character under the cursor
CTRL E	Cursor moves to the *end* of the line
CTRL F	Cursor moves one space *forward*
CTRL N	Cursor moves to the *next* line
CTRL O	Line is *opened up* above the line with the cursor. Be sure cursor is at the beginning of the line
CTRL P	Cursor moves to the *previous* line

Try to associate each control character with a meaningful word so that you do not have to refer to the manual frequently.

Practice Activities

Change some of the lines in the PRESENT procedure so they match the following program. Use the editor and control characters above.

```
TO PRESENT
FD 20
RT 45
FD 35
RT 45
FD 20
RT 45
FD 35
BK 17
RT 45
FD 20
BK 10
RT 45
FD 17
BK 35
END
```

For more practice see Chapter 6.

INITIALIZING A DISK

Preparing or Initializing a Disk to Save Your Programs

Follow these instructions to initialize or prepare a blank disk to *save* your programs:

1. Turn off the computer (the switch is on the back left) and insert the 3.3 DOS System Master disk into the drive. Turn on the computer.
2. When the light on the drive goes off, remove the disk.
3. Insert your blank disk into the drive. Type INIT HELLO and press RETURN. The drive will run for nearly a minute while it prepares your disk.
4. Type CATALOG and press RETURN. You will see that HELLO is now present on your disk. If not, repeat the procedure from step 1 above.
5. Remove your disk and insert the Logo disk. Press RETURN.
6. Even though the message on the screen tells you to insert your disk at this time, *do not do it the first time you are using the new disk*. Press RETURN again.

7. When you see the message WELCOME TO LOGO, now is the time to insert your new disk. After doing so, type SAVE "STARTUP "AIDS and press RETURN. This is necessary the first time you use your disk because it places a library of prewritten procedures on your disk.
8. Again you receive the friendly greeting

```
WELCOME TO LOGO
?
```

and you are ready to write a procedure.

How to Save Your Programs

If your new disk is not in the drive at this time, follow these steps:

1. Turn off the power, insert the Logo disk. Turn on the power.
2. When the light on the drive goes off, insert your initialized disk and press RETURN.
3. Wait until you receive the message

```
WELCOME TO LOGO
?
```

Now you are in business.

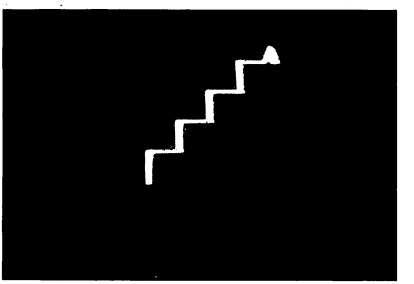
SAVING

Saving One Program

Here are the steps to follow in order to *save* a program. Follow these steps *only* if this is the first procedure you have entered after loading Logo into your computer.

1. Write a procedure:

```
TO STAIRS
CS
REPEAT 4[FD 20 RT 90 FD 20 LT 90]
END
```



2. Look for the message

```
STAIRS DEFINED
```

3. Type

   SAVE "STAIRS

4. STAIRS is now saved. You will see the message

   1 PROCEDURES SAVED

   If this was not the first procedure written after loading Logo, more than one procedure will be indicated as saved.

Checkpoint: Turn off the power and reload Logo into the computer. Insert your disk and press RETURN. To see if your program was really saved, type

LOAD "STAIRS

5. After the drive light goes off, type STAIRS and your procedure should be drawn on the screen.
6. Type the command CATALOG and you will see all the procedures you have saved. Since this was your first one, you should only have HELLO and STAIRS. It is not necessary to turn off the computer each time as in the CHECKPOINT above. Only the CATALOG operation is needed.

Saving Several Procedures

If you are defining several procedures at one sitting, *do not save each one separately!* When it comes to saving more than the first procedure, Logo does something weird. To explain this statement, consider this hypothetical situation: Logo is loaded, and the following procedure is entered:

```
TO LINE1
FD 100
END
```

If you typed SAVE "LINE1, you would receive the message

1 PROCEDURES SAVED

If you then entered a second procedure:

```
TO LINE2
BK 100
END
```

and typed SAVE "LINE2, you would receive the message

2 PROCEDURES SAVED

Each time you write and save a new procedure, every other procedure you have written up to that point will also be saved under the file name you place after the word SAVE. It is not efficient to have every procedure saved over and

over again. However, this will continue to occur with each subsequent procedure you enter and save, but there are two ways to get around it.

Erasing Procedures

One way is to erase all procedures in the memory before saving a new one so that there is nothing in memory that you don't want saved in that file. In other words, since the computer saves *every single procedure in memory* as the result of one SAVE command, then just erase what you don't want saved. Be sure that if you wanted other procedures saved, you would already have done so before erasing the procedures in memory. Here is a situation where you want to clear the memory, but save specific procedures:

1. You have written procedures named CHAIR, TABLE, and LAMP.
2. You saved them with the command

```
SAVE "ROOM                (CHAIR, TABLE, and LAMP
                           are in this file now)
3 PROCEDURES SAVED        (Screen output)
```

3. Now you write procedures named FLOWER, VASE, and BEE. Before you attempt to save them, you must omit the other three procedures, under the name ROOM. Use ERASE (or ER for short). Enter the command

```
ERASE [CHAIR TABLE LAMP]
```

4. If there is more than one procedure to erase, the names must be listed inside the brackets, separated by a space. If only one procedure is to be omitted, it can be in this form:

```
ER "CHAIR
```

5. The three procedures named in the ERASE command (CHAIR, TABLE, and LAMP) are now gone from the current memory but, of course, are still saved on the disk under the file named ROOM.
6. Now the SAVE command can be given:

```
SAVE "PHOTO               (User input)
3 PROCEDURES SAVED        (Screen output)
```

PACKAGE Command

The second alternative to avoid saving procedures on top of procedures is to use a command called PACKAGE. It is like placing certain procedures in a special package with a special name. For example, if you had written procedures

named TABLE, CHAIR, and LAMP, you could enter these lines:

```
PACKAGE "ROOM[TABLE CHAIR LAMP]
```

Inside the package named ROOM, the procedures TABLE, CHAIR, and LAMP would be stored and no others. After packaging these procedures, you still must save them, but a special format is necessary:

File name	*Package name*
SAVE "ROOM	"ROOM

It is OK to use the same two names for the file to be saved and the package. The command above saves in the file named ROOM whatever is in the package named ROOM. The file name in which you wish everything to be saved comes first, preceded by a quotation mark, and the package name comes second, also preceded by a quotation mark. When using packages to store your procedures, it is not necessary to erase any procedures or to worry about piling up all your programs.

Note: It is possible to save the same procedure in more than one package. For example, let's say that you want TABLE to be saved in the file named ROOM as well as in a file named PHOTO. Just name TABLE in each package you want to include it in. Then you decide that you want to change the TABLE procedure in the ROOM file but leave it the same in the PHOTO package. This is how it is done:

1. Erase all procedures from the workspace (memory)—assuming ROOM and PHOTO have been saved. The command to use is ERALL.
2. Load ROOM and call TABLE into the Logo editor:

   ```
   LOAD "ROOM
   ED "TABLE
   ```

3. Make the changes you wish, type CTRL C to exit the editor, and enter

   ```
   ERASEFILE "ROOM
   SAVE "ROOM
   ```

The ERASEFILE command eliminates the old file ROOM and then saves it once again with all the same procedures plus the edited TABLE. TABLE was not edited as part of the PHOTO file, so it will remain in its original state. If you desire to have both TABLES edited in the same way, you

would have to follow the above three steps for each TABLE you want changed in each package it appears.

POTS, POPS, PO

If you ever want to check to see exactly what procedures you stored under a certain file name, use these commands:

```
ERALL          (To erase all procedures in memory)
LOAD "ROOM     (Loads just the procedures in ROOM)
POTS
```

The PRINT OUT THE TITLES command (POTS for short) will cause a listing of each procedure in memory—in this case "ROOM. If the ROOM file holds TABLE, CHAIR, and LAMP, the screen output resulting from the above line would be:

```
TO TABLE
TO CHAIR
TO LAMP
```

Another command, PRINT OUT (PO), will list each procedure named, if it is in memory, along with the name of the program instruction lines. It is necessary to specify certain files:

```
PO "TABLE          (TABLE procedure will be output)
PO [TABLE CHAIR]   (TABLE and CHAIR procedures
                    will be output)
```

When you want to see the procedure instructions for procedures in a certain package, the following command is used. Here ROOM is a package name.

```
POPS "ROOM      or      POPS [TABLE CHAIR]
```

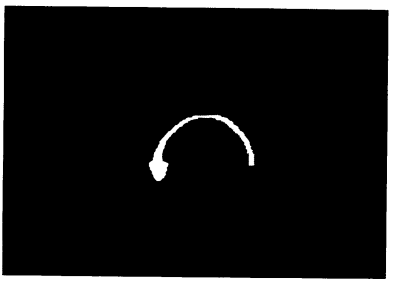
A Trip to the Library

The prewritten programs mentioned earlier are in the STARTUP file. They are on the Logo disk and also on your initialized disk if you followed the instructions on page 126. They are really just shortcuts that are available and include seven different files. Six will be presented here. Only two of them have abbreviated forms.

```
ARCLEFT      or      ARCL (Short form)
ARCRIGHT     or      ARCR (Short form)
CIRCLEL
CIRCLER
ARCL1
ARCR1
```

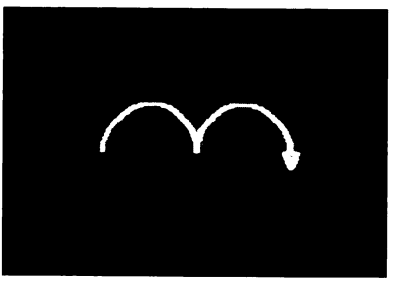
All of the procedures deal with producing arcs and circles. They require inputs—numbers that help define what kind of circle or arc to draw. The following procedure means to draw an arc to the left with a radius of 30 turtle steps and 180 degrees around.

```
CS
ARCL 30 180
```



This will produce a half-circle (since 180 degrees is half of 360 degrees, a complete circle; the circle is 60 turtle steps across and the radius is half of the diameter).

```
PU
HOME
PD
ARCR 30 180
```



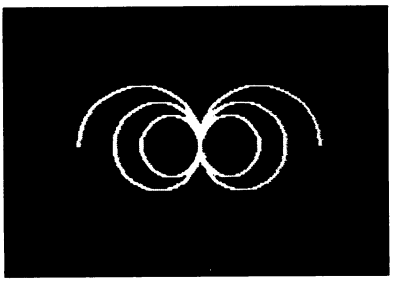
This is a mirror image of the first line, with the arc going to the right, using identical radius and degrees. Why are PU, HOME, and PD used here? Perhaps you will discover it as we go along.

The CIRCLEL command needs only one input number, the radius, since it will automatically be 360 degrees or one revolution.

```
PU
SETPC 5          (Add some color for fun)
HOME
PD
CIRCLEL 20
```

Add the mirror image of this simply by entering these lines to the previous ones:

```
PU
HOME    PD
CIRCLER 20
```



The other two commands, ARCR1 and ARCL1, require still different inputs.

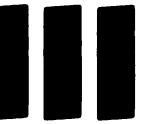
```
PU
SETPC 1
HOME
PD
ARCL1 4 36
ARCR1 4 36
HT
```

The first number (4) represents turtle steps, and the second number (36) refers to the number of times you want the turtle to go 4 steps. The turtle automatically turns 10 degrees before each new step size is executed. Since 10 degrees times 36 is 360 degrees, we can expect a circle again. Predict what would happen with the input numbers of 10 and 18? Try it.

Practice Activities

1. Using different pen colors, draw several circles inside each other using CIRCLER and CIRCLEL and connect them all at one point.
2. Draw concentric circles—ones that are not touching each other, using CIRCLER and CIRCLEL.
3. Draw what resembles bug eyes with antennae. Use CIRCLER and CIRCLEL, ARCR and ARCL, and ARCL1 and ARCR1.

AN OVERVIEW OF LOGO



Logo is a unique programming language enjoyed by people of all ages. From preschoolers in Texas to graduate students at MIT, Logo offers adventures, challenges, and discoveries to all who partake in it. Now that you are familiar with how the turtle operates, perhaps you are interested in learning about its personal life—where it came from, what's being done with it, and possible future directions.

COMPUTER. The very word intimidates many of us. For parents and teachers who are deeply committed to preparing children for the future, computers can be especially frightening. Because of the rapid and relatively recent growth of the computer industry, it is only natural for us to feel a little apprehensive about the capabilities and desirability of computers. Being unfamiliar with computers, we envision an impersonal future where computers will run the lives of today's children and human contact will be minimal. In this future world, we sometimes fear, human beings will become machinelike servants of an almighty computer.

Fortunately, there is one major fallacy in such thinking: human beings are the creators and programmers of computers, not their victims. True, we hear horror stories about computers replacing teachers and other professionals which seem to proclaim that computers possess unlimited power. In reality, computers cannot explain anything for they cannot think, or can only think in a relatively primitive way. They can do no more than to process information given to them by human beings.

Keeping in mind the fact that computers are created by human beings for our own purposes, we can begin to see that a computer should be no more intimidating than other types of machinery that we use every day. Computers are simply able to serve us in a greater number of ways than can most machines. Consequently, it is more realistic to think of them as valuable rather than as intimidating. Since it certainly seems that computers are going to be around for a long time to come, it is necessary for today's children to learn to feel comfortable around computers and to develop a sense that computers are helpful, not threatening. Furthermore, since computers are available to us today, it is only sensible to familiarize ourselves with them and work to discover ways in which they can be used to improve the quality of life.

One of the most obvious and vital ways for computers to be used now and in the future is for educational purposes. A computer can also serve as a teaching tool, but its function is more valuable as a tool of learning. A child's first experiences with a computer can easily determine whether he or she will view the computer as helpful or intimidating. For children to feel that computers are approachable, that they are machines rather than frightening beings, children must have early and frequent contact with computers.

DEFINITION, ORIGINS, HISTORY

A Definition of Logo

Seymour Papert, a professor at the Massachusetts Institute of Technology, has created a computer language called Logo, which allows the computer to become accessible to children as young as three years of age. Papert's vision is to have the child teach the computer and thus experience "what it is like for mathematics to enable whole cultures to do what no one could do before."[1] Papert envisions a future in which every child will own a computer. He created Logo in hopes that eventually children will become as comfortable with computers as they are with pencils.

Logo's accessibility to children derives in great part from its simplicity. The language consists of basic English words, such as *right*, *left*, *forward*, and *backward*. Appropriate abbreviations for such instructions make the language easy for very young children to understand and use. To carry out Logo commands, Papert invented the turtle, "an object to think with."[2] Later, sprites and demons were introduced into the Logo environment to make the system livelier, more complex, and more appealing to children.

The Origins of Logo

In his book, *Mindstorms*, Seymour Papert discusses the earliest roots of his educational theory, drawing on experiences from his own childhood. Papert explains that before the age of two, he had become interested in automobiles. He developed a particular fascination with—or, as he terms it, love for—gears, and to this infatuation he attributes the development of his unique educational theory.

Papert describes his early attempts at constructing gear systems with an erector set. Often he turned imaginary gears in his head. When he was old enough to enter a traditional educational system, he was able to adapt this habit of visualizing gear systems to help him solve mathematical problems. Papert stresses the point that, without the intervention of adults, he had developed a way of thinking about mathematics merely from observing and experimenting with gear systems. This led to his fundamental concept of education: What and how children learn is directly influenced by the materials and models available to them.

---

[1]Seymour Papert, *Mindstorms: Children, Computers, and Powerful Ideas* (New York: Basic Books, 1980), p. 74.

[2]Papert, *Mindstorms*, p. 55.

Another of Papert's influential experiences was his study with epistemologist Jean Piaget at the Center for Genetic Epistemology in Geneva from 1959 to 1964. Piaget's theories of learning had a profound effect on Papert's own theories and indeed serve as a springboard for Papert's thinking.

In simplest terms, Piagetian learning may be defined as "learning without teaching."[3] The underlying assumption of Piaget's thought is that a child is capable of determining his or her own process of education. Rousseau and Smith note that "the crux of Piagetian thought is that a child learns through experience and develops a framework for dealing with his or her environment in relatively predictable stages."[4] Papert is concerned with Piaget's notion of spontaneous learning by the individual encountering and manipulating the environment. This notion is in direct conflict, Papert asserts, with the idea of rigidly prescribed curricula found in traditional school settings.

Papert stresses the importance of the materials that a culture provides children for their education. Using his own childhood to illustrate this, Papert notes that the presence of gears in his environment provided him with an intellectual structure for solving problems and a way to think about mathematics. He finds this experience lacking in the traditional classroom.

He also emphasizes the affective aspect of his childhood experience with gear systems. He did not merely experiment with the gears; he fell in love with them. This affinity gave him confidence and delight in the experience of learning, another quality he finds sadly deficient in the traditional school setting.

Papert's own childhood, his study with Piaget, and his years of work in research and development of educational technology at the Massachusetts Institute of Technology laboratories resulted in the development of Logo as a viable alternative to traditional education. Papert developed what he refers to as a Utopian educational system in which children find learning simple, natural, and fun. The basis of Papert's brave, new vision is his definition of successful learning.

---

[3]Papert, *Mindstorms*, p. 52.

[4]Joseph F. Rousseau and Stephen M. Smith, "Whither Goes the Turtle?" *Microcomputing* (September 1981): 52-55

History of Logo

With the help of colleagues and associates at MIT, Papert set out to design a computer language that was suitable for use by children and yet had the power of professional programming languages. The name Logo was chosen to indicate that the language is "primarily symbolic and only secondarily quantitative."[5] Logo was first implemented in 1967, funded by grants from the Educational Technology Group at the research firm of Bolt, Beranek, and Newman.

During the 1968–69 academic year, Logo was introduced to a group of twelve seventh graders attending Muzzy Junior High School in Lexington, Massachusetts. It was used on a DEC PDP-10 computer. At that time, Logo did not possess graphics capabilities. Students used Logo to write programs that translated English into Pig Latin, to play games of strategy, and to generate "concrete poetry."[6]

The following year the Logo Project was established in the Artificial Intelligence laboratory at MIT. In an attempt to make Logo more appealing and accessible to younger children than the junior high school children who had experimented with it, Papert invented turtles. After more than ten years of testing and development, Logo is indeed more attractive to children because of its emphasis on color, graphics, and simplicity.

In describing how users learn to "speak" Logo, Papert uses the analogy of children learning to speak any language. They learn their native languages quite spontaneously, a fact that aided in the development of Piaget's notion of learning without teaching. Learning a foreign language is another matter entirely. Papert notes that the average child has great difficulty in learning a second language in a school setting. A child who is trying to learn French, for example, is likely to experience a good deal of frustration because of the way languages are taught in the classroom. Send that same child to France, Papert suggests, and the child will learn French almost as naturally as he or she learned his or her native tongue. By interacting with an environment in which French is spoken, the child will invent his or her own way of learning the language.

Papert advocates the creation of an analogous natural environment in which the "language" of Math is spoken—Mathland. Children who are familiar with such an environ-

---

[5]Papert, *Mindstorms*, p. 211.

[6]Papert, *Mindstorms*, p. 218.

ment develop a language and, consequently, an intellectual structure for mathematical processes. Logo and turtles make up this environment. The child speaks in Logo to the computer, causing the turtle to perform what are in effect mathematical formulae. Thus, language and mathematics, traditionally split into opposing camps, are wedded in the child's learning experience. The child is given a model upon which to base increasingly complex and sophisticated knowledge.

Again, given the proper environment, this learning occurs very naturally and painlessly:

> Computers are the Proteus of machines: they take on many different forms. One of their manifestations is as mathematic-speaking beings. If children grew up surrounded by such beings, the learning of mathematics might very well be much like the learning of a spoken language.[7]

Children actually require very little prerequisite knowledge to enjoy the Logo experience. They must recognize letters of the alphabet and be able to locate them on the keyboard. They must be shown how to operate the computer—how to turn it on and off. And they must understand a few basic concepts such as up, down, left, right, forward, and backward in regard to their own movements.

THE LANGUAGE OF LOGO

Definition of Turtles

There are two basic types of turtles: floor turtles and light turtles. *Floor turtles* are dome-shaped, have wheels, and can be handled like any mechanical toy. Equipped with a pen, the floor turtle draws upon the floor as it moves in accordance with commands entered on the computer keyboard. The *light turtle* appears on the video screen as a triangular-shaped light which responds to commands by moving about appropriately on the video screen. Like the floor turtle, it leaves a line as it moves.

Beginning students are often introduced to Logo with the floor turtle. It is sometimes easier for young children to grasp the meaning of a certain command they give when they are able to see the floor turtle's response before their eyes. For example, a child who wants the turtle to make a circle can get up and move in a circle, analyzing the movements until he or she sees that to make a circle means to move forward a bit, then turn a bit, move forward a bit, then turn a bit, and so forth, until the circle is complete.

---

[7]Papert, *Mindstorms*, p. 232.

Because the floor turtle is in the same physical space as the child, the child often finds it easier to give appropriate commands.

The light turtle operates in the same way but on the video screen. Logo users learn to give instructions to, or program, the turtle by teaching it new words. For instance, if the child has gotten the turtle to make a circle through the proper series of movements, he or she can name the shape CIRCLE (or CR) and from that time on, the turtle will make a circle upon receiving that command. As a result of teaching the turtle to make a circle, the child has discovered something about geometry. The computer has not taught the child; rather, the child has used the computer as a learning tool. The child is both teacher and learner in this situation, and the motivation has been to create a shape on a video screen, a shape which he or she can instruct the turtle to recreate at any time.

The Functions of Turtles

With Logo, the user gives the turtle commands such as RIGHT or FORWARD, but it is also necessary to give more specific information. For example, to simply direct the turtle forward gives no indication of how far forward. To direct the turtle to go right does not say whether it should make a sharp turn or a slight curve. The student controls the moves by entering numbers. FORWARD 1 is a shorter distance forward than FORWARD 50. Similarly, RIGHT 3 is a less extreme angle than RIGHT 60.

All turtles have at least two properties in common: *heading* and *position*. Stated simply, the turtle faces in one particular direction (heading) while "standing" at one particular point (position). These two properties help the child to identify with the turtle. Like a person, or a house, or an animal, the turtle must stand in a particular spot and face a certain direction. Thus, the turtle has mathematical properties which are concrete and easy for the child to conceptualize.

As the child discovers more about such mathematical properties, he or she can program the turtle to possess more complex properties, controlled by teaching the turtle more words. For instance, turtles, called *Dynaturtles*, can take on properties such as velocity and mass. Turtles can also take on the property of acceleration. It is quite possible, and even likely, that a child will begin to discover Newton's laws through working with such turtles. Students can then

teach the turtle whatever properties they discover in working with it.

Other exciting possibilities exist in the Logo turtle microworld. For example, there may be more than one turtle in a microworld just as there are often several people in the child's world or several molecules in a substance. The turtles may be independent of one another, or they may be linked. If they are not linked, the turtles can be commanded individually by giving them names, such as turtle 1 and turtle 2. In a microworld, it is also possible to program the turtles to take commands from each other. For instance, in a "mirror Turtle" microworld, the child programs the turtles in such a way that when the turtle is given a command, its partner does exactly the same thing.

Because the student creates and discovers many possibilities in this microworld, he or she stays interested and keeps learning, using the turtles for his or her own purposes and motivating an inner desire to continue the creation of a more complex and challenging microworld.

CHILDREN AND LOGO

Case Study

So far, Logo sounds relatively simple, but how does it work in practice? The Lamplighter School, a private school for children of preschool age through the fourth grade in Dallas, Texas, has served as a testing ground for Logo. A number of teachers without special training in computer science began learning Logo on personal computers. In 1979, these teachers trained their colleagues at the Lamplighter School and implemented programs for teaching Logo on microcomputers in their classrooms. While Lamplighter students used the microcomputers daily, they also received traditional lessons in art and mathematics.

Preschool-aged children used Logo to discover concepts of shape, position, color, direction, and speed. Children in primary grades were taught to use simple Logo graphics commands. The children actually discovered, rather than learned. For example, one child found that a square consists of four equal sides, while another discovered that a rectangle has two sets of two equal sides.

The children at Lamplighter School were extremely innovative in their applications of Logo capabilities. One third-grader produced original dramas in which the text was printed across the bottom of the screen while sprites and turtles acted out the drama's plot. Another third-grader re-created a space shuttle landing using Logo. EXPLODE

is a program written by Lamplighter students in which thirty-two balls of different colors spray outward from the center of the screen in a circular form, return to the center of the screen, and then repeat the performance. In another such program, RAINBOW, sprites change color continually.

Studies of Lamplighter children six months after implementation of the Logo system demonstrated that the children had acquired several new skills. They were able to use subprocedures; that is, they were able to analyze and utilize various component parts of complex programs. They had also discovered recursive programming. *Recursive* programming refers to including the name of the program within the program itself so that it can be run again continuously.

Experimenters were impressed by a number of other features of Lamplighter School's experience with Logo as well. For example, the absence of pressure imposed on students by the threat of grades allowed students to progress at their own rates and significantly decreased competition among the children. Children of all ages and levels of ability were able to enjoy a feeling of success.

The children served as their own teachers. They felt a sense of control over their learning environment and process. Since the microcomputers were dependent on the children for instruction, the children suddenly found themselves in the position of choosing to give or to withhold information.

Children took turns in the role of class leader. They admired, praised, and implemented each other's programs. Each child created unique, individualized programs and they enjoyed the chance to share this work with classmates. Hence, all of the children at Lamplighter School felt themselves to be important and worthwhile members of the class.

Pat Mattingly, director of Lamplighter School, noted that even the three-year-olds who participated in this experience were not at all frightened or intimidated by computers. No one had presented the computer to the children as a threatening entity. Obviously, children can begin to become computer literate at younger ages than has previously been considered possible.

While the Lamplighter School experiment highlighted many benefits of Logo, it was still a long way from Papert's ideal educational environment. He envisions an educational system far different than any we know today, a system that will prepare children for living comfortably and knowledgeably with computers on a daily basis.

Children as Teachers

According to Papert, learning is successful when it occurs as spontaneously as when a child learns to speak his or her native language. The learning must be accessible to all children, not to the intellectually or economically elite only. It must include none of the rigid, inhibitive aspects of traditional "school learning": tests, deadlines, curriculum, grades, or teachers.

The new educational system must provide its students with the freedom they require to direct and control their own processes of learning, and it must also provide the materials with which they can do so. Papert notes that in an age that increasingly demands computer literacy, schools must provide children with the opportunity to interact with computers. Many school systems have done just this by implementing programs of computer-assisted instruction.

Most computer-assisted instruction programs have been geared toward using computers for drill and practice much as teachers have traditionally done. Papert describes this process as the computer programming the child. The computer merely provides routine drills, reinforcement, or remediation. In Papert's view, the child is too passive in this system.

The theoretical underpinnings of Logo insist that the child take on a more active role in learning with the computer. Using Logo, the child begins to program the computer, rather than being programmed by it. This gives the child the power to use the computer as a tool that he or she has chosen for himself or herself in the learning process. In this way learning becomes self-directed and self-controlled, allowing the child much more opportunity for creative experimentation and satisfaction from the experience.

Because the computer still seems like a mysterious, omnipotent, and complicated machine to many, some people fear that children are incapable of learning to program computers effectively. Papert dismisses this misconception:

> To program a computer is merely to communicate in a language understandable to the programmer and computer. Since all children are capable of learning languages and of talking, all normal children are capable of learning to program a computer.[8]

If computers are introduced to children at a young enough

[8]Papert, *Mindstorms*, pp. 5–6.

age to avoid being influenced by the popular fear of computers, they will have no disabling prejudices to overcome in working with these educational machines.

Children as Epistemologists

Perhaps one of the greatest benefits of a Logo system is that it causes children to become epistemologists for themselves. They are put in the position of teaching the computer to think, and this necessitates developing insights into the way that humans think. As children examine the ways in which they think, they gain insight into themselves as people, about the way tasks can be analyzed and performed, and about the world around them. They find ways to build their own intellectual structures and begin to become their own teachers.

Once children have taught the computer how it must think in order to follow the instructions it receives, they must formulate written commands. This increases children's expertise in commanding. It also allows them the pleasure of exercising their wills in a world in which they are very often at the mercy of adults. Using their ability to give orders and watching the turtle obey their commands gives them a sense of their own capabilities. It gives them a sense of power that they rarely experience in other facets of their lives. This leads to an increased sense of identity, volition, and independence.

A closely related benefit derives from the fact that before a child can learn to control the turtle, he or she must learn to speak Logo. The child learns that words result in actions. In this situation, Logo commands result in an exciting visual experience over which the child has control. This develops the child's respect for and interest in the language.

Children who have worked with Logo have been found to be more articulate in discussing program bugs and asking for help in correcting them. In effect, they develop a jargon that they can use to describe processes and errors so that they can be easily understood. When the children become more articulate about their Logo experience, the adults charged with the responsibility of helping them require less professional training.

Some people object to Logo because they feel that computers hamper the development of reflective thinking. They are concerned that children especially may come to see themselves as part of the hardware of a computer system

rather than as a controlling human being. Papert does not believe this to be a valid viewpoint. He notes that he has seen children, engaged in work with Logo, thinking and speaking with frequent references to themselves. They do not seem to lose sight of themselves as the human beings who regulate and manipulate the Logo environment.

Another objection to Logo learning is that working with computers may make children too analysis oriented and too verbal in their thinking. Papert answers this objection through a technical analysis of the psychology of learning, and his discussion would be useful for educators who are considering implementing a Logo system.

LOGO AS AN EDUCATIONAL TOOL

Although Logo has the potential for important uses in the home, we are most likely to find it in the classroom where it has many varied functions. Educators have proclaimed it useful in supplementing traditional classroom learning, improving and even revolutionizing the classroom, and developing the mathematical, artistic, and writing skills of young students.

Because of its versatility, Logo can easily supplement lesson plans and traditional curricula. For example, in a classroom with one computer, children can take turns, each having access to Logo twice a week for a specified amount of time. Along with improving the skills which children are being taught in the classroom, Logo will provide other valuable opportunities. The excitement of using skills on the computer can motivate the generally apathetic student to pay closer attention to his or her lessons. Logo also offers students a chance to work independently and to create programs which no one else has discovered. Another advantage of supplementing traditional learning with computer learning is that the child gets some exposure to a machine which will probably become more and more important in the world around him or her. The opportunity for a student to progress at his or her own rate is another useful Logo feature.

Although Logo can be used very effectively to supplement traditional learning, Papert has a more all-encompassing plan for its implementation. He dreams of using the new technology not merely to improve schools but to create a better alternative. He believes that children can benefit more from discovering with the computer than they ever will

through being forced to memorize and work according to an arbitrary traditional curriculum. According to its creator, Logo can do more than merely supplement traditional classroom learning—it can revolutionize it.

Turtle Geometry

Turtle geometry is the best known and most widely used function of Logo. The turtle functions as a bridge between body geometry and formal geometry. That is, because the turtle makes concepts such as lines, angles, and quadrants concrete, the child can relate geometry to his or her own experience.

Turtle geometry is also useful for building a hierarchy of knowledge. For example, Pamela taught the computer to draw a square and a triangle. Eventually, it occurrred to her that she could make a house by placing the triangle on top of the square. She had to acquire new knowledge and skills in order to position the triangle directly upon the square in the proper spot.

When a student is interested in solving nonconcrete, abstract, geometrical problems, Logo computers provide a nongraphic mathematics mode for problems ranging from the very simple to the highly complex.

Logo in Art

The graphics mode of Logo serves to help the child to develop creativity and artistic skills. Children can construct shapes limited only by their imaginations. They can experiment with proportions, line thicknesses, and color. To create dynamic screen pictures, speed and direction properties may be used. This mode is particularly effective in giving the child an awareness of his or her own creativity, as it is very likely that the child will design pictures and moving objects that no other child has ever created. The child can store this procedure and build upon it during future sessions with Logo. Thus, Logo provides not only academic opportunities, but artistic ones as well.

Logo and Social Skills

Children actively involved in the Logo experience frequently find themselves communicating more often with peers as well as with teachers. Although they usually work alone with the computer, they are eager to meet and talk with others engaged in similar activities in order to share their experiences. They have common excitements, frustrations, and interests to share with other children. This in-

creased interaction leads to enhanced social skills that carry over into other areas of the children's lives. They learn that sharing experiences with other people is interesting and worthwhile.

An especially attractive benefit of implementing a Logo system is that it produces a greater tolerance of errors in its users. The atmosphere of a Logo environment is conducive to learning by trial and error. Everyone, including the teacher, is a learner with each new activity, each new program written. Mistakes are expected, not punished. This creates a far less threatening environment than the conventional educational setting in which children are expected to learn by demonstrating perfection on an examination in order to succeed.

Not only are errors expected in a Logo environment, but they come to be valued. They give children a way to see cause and effect. Errors not only indicate a problem, but help to show how programs can be corrected and strengthened. A mistake is not cause for shame, but an intriguing opportunity for growth and change.

Consider, for example, a traditional classroom in which children are tested to determine whether or not they can perform mathematical calculations. The testing tool is generally a written examination. Children feel anxiety and frustration when they are asked to do pages of problems that seem to have no relevance or interest to them. The teacher collects the examination papers, marks errors, scores the papers, and returns them to the students. The students are often humiliated by mistakes and are very rarely given the opportunity to correct or learn from them. Naturally, students try to forget their errors as quickly as possible so that they no longer need to feel embarrassed about having performed inadequately.

The Logo environment is far less threatening to children. Debugging, or correcting, a program is not some deviant behavior to be avoided at all costs; it is an inherent part of the process of programming the computer. Carefully examining mistakes leads to the discovery of viable solutions. Children often find better ways of accomplishing tasks. From this they learn that there are degrees of correctness and incorrectness. The process also becomes a more important learning experience than the product.

Another feature of the Logo system is that programming in Logo takes on a new concreteness for children. Procedures created by the students become, in Papert's

terminology, "manipulatable entities that can be processed and handled in numerous ways,"[9] naming, storing, retrieving, altering, and combining them to make larger processes.

Logo in Special Education

Research has shown that Logo is effective in teaching exceptional children. This is especially obvious with gifted children. Since Logo is a self-paced system of learning, the gifted child is not limited by lesson plans or slowed down by other children in the class. Furthermore, learning to use Logo well will make it easier for the child to learn more difficult and complex computer languages later in his education.

Logo's features also lend themselves to use by educationally handicapped children. Papert's research has shown that controlling the turtle is an absorbing pastime for mentally retarded and autistic children. It also provides highly successful learning experiences for children with physical handicaps. Some of the Logo features which make the language productive as a learning tool for handicapped children include its emphasis on process rather than product and its exercising of the hands and fingers—a side effect of typing on the computer keyboard.

Often, handicapped children lack self-confidence and have a low self-image, especially in academic matters. This is hardly surprising in light of the fact that the physically handicapped child often spends much time unsuccessfully trying to do what other people want him or her to do.

Sylvia Weis has studied applications of Logo in the special education environment. She writes about Keith, who has a severe reading disability known as dyslexia, and who was introduced to Logo at age eleven. He had attended a special school for four years and had been labelled as having a poor attention span and a low level of motivation. After forty minutes with Logo Keith was controlling the turtle better than most beginners of any age. He created a program which drew a snowman and thoroughly mastered the Logo commands necessary to do so. At the end of forty minutes, Keith did not want to stop working.

The Logo system appealed to Keith for two major reasons. First, he was in control of his own actions, and thus he decided what he wanted to do rather than passively following another person's instructions. Second, Keith was

[9]Papert, *Mindstorms*, p. 223.

able to use a talent; he used skills not traditionally valued in an academic setting where superior spatial skills are not deemed as important as language skills.

Logo has also been valuable in the education of students with cerebral palsy, a movement disorder caused by a lesion in the brain which occurs before the child's second year of life. Mike has cerebral palsy which affects both legs and both arms and seriously impairs his speech. He is confined to a wheelchair, and prior to his experience with Logo, was dependent on others for his own learning. At age seventeen, Mike began working with Logo and is now a competent computer programmer who has been accepted as a Computer Science Major at the Boston campus of the University of Massachusetts.

Logo permits students like Mike to perform liberating functions which, without Logo, would require outside help. With Logo, such students can use the computer to "write down" homework problems rather than doing them mentally. They can do language arts assignments along with music theory and composition. Logo provides children with the opportunity to do spatial problem solving, contributing to academic success and an improved self-image. The potential uses of Logo by exceptional children seem almost unlimited.

Logo in Adult Education

In *Mindstorms*, Papert states clearly that Logo is not a toy or a language for children alone. Some versions contain very sophisticated functions such as parallel processing and message passing. Logo's list structure allows for the production of lists of lists, which makes it possible for Logo procedures to construct, modify, and run other Logo procedures.

The simpler functions of Logo make it an ideal language for beginning adult users with little or no prior mathematical knowledge. Once the beginning adult has become familiar and competent with Logo, his or her knowledge will easily transfer into the learning of more difficult computer languages. The turtle itself has been adapted as a means for teaching other languages. For instance, the turtle is currently being used at the college level to teach PASCAL, a higher level programming language. This method of learning the language makes use of commands which were originally developed in Logo.

LOGO IN THE HOME

Children and Adults

One of the most attractive features of Logo is its versatility. A home system with Logo is advantageous for both parents and children. Its nongraphic math mode makes it helpful in working with the household budget. Its language capabilities can be used for keeping lists and leaving or receiving messages. Logo in the home will allow parents to become more familiar with computers and thus to participate more fully in their child's education, especially since computer-aided education is becoming more and more common.

Before parents can help their children to adjust to interacting with computers, they themselves must become more comfortable with them. Communication between parent and child will be enhanced through sharing of experiences, problem-solving sessions, and increased interactions.

For those who set up Logo environments in their homes, there are other benefits besides those enjoyed by educational institutions. Logo offers parents the opportunity to supplement the education that their children are receiving in traditional school settings. Since it will take time for schools to implement programs like Logo, parents can give their children an early start in learning to work with computers, a skill that will be vital to the adults of tomorrow.

Not only does Logo encourage parents to participate in their children's education, but it also constitutes an activity that parent and child can engage in together. This sharing will enhance the relationship between parents and their children by giving both a common experience. Parents and children who share the Logo experience will find new avenues of communication opening to them.

Logo is also an attractive alternative to other less productive activities on which children may spend their time. A child who is learning to create graphics on a video screen is spending his or her time more productively than a child who is watching television. With increasing concern about the effect of television on the cognitive development of impressionable young minds, parents will find educational tools like Logo especially appealing.

Another feature that parents of school-aged children are likely to find attractive is Logo's adaptability for video games. As preprogrammed, prepackaged video games and video game arcades become more and more popular, parents are beginning to express concern about the amount of time and money that their children invest in what often seem like

mindless games. By implementing a Logo system in the home, parents can encourage their children to turn this interest into a creative activity. Logo allows children to compose video games rather than merely playing games that have been marketed widely for public use. Children can create their own games that reflect their interests and hobbies, and parents can feel that game playing is a worthwhile activity.

MORE ABOUT LOGO

Other Modes of Logo

Other modes in the Logo environment provide further possibilities for learning. Depending on the computer company which produces the Logo software, some of the capabilities discussed in this book may or may not be available for a particular computer. One of these is the *sprite mode*. Sprites are invisible "spirits" that transport standard shapes, or creations, made by the computer user. Once a shape has been formed, any or all of the sprites may be commanded to carry it about on the screen. This can produce a dynamic arrangement on the screen. This naturally leads to an appealing opportunity for the creative expression of the child.

Another mode is the *demon mode*. Eyster describes the demon as a creature which "drifts like a whistle-bearing referee about the screen, measuring distance, watching quadrants, checking conditions."[10] This mode makes higher level geometry and algebra discoveries possible.

There is also a nongraphic mathematics mode for working out traditional mathematics problems as well as an editing mode that makes word and list operations possible. The fullscreen procedure editor supplies an opportunity for improved writing skills, as has been discussed earlier. Children often prefer working on a computer to working with pencil and paper, and the edit mode often encourages students to work harder on their writing skills. The various modes offered in Logo fulfill practical as well as theoretical educational goals.

The Nuts and Bolts of Logo

The necessary hardware for Apple Logo includes an Apple II or Apple II Plus computer with 64K bytes of memory and a disk drive. (See the following table of Logo version vendors.)

---

[10]Richard H. Eyster, "Seymour Papert and the Logo Universe," *Creative Computing* (December 1981): 70–74.

Seven Major Versions of Logo

Logo Version	*Hardware Required*	*Manufacturer*
ATARI	ATARI® 400™ or ATARI® 800™	Atari, Inc. P.O. Box 427 Sunnyvale, CA 94086
Commodore	PET 64™	Commodore Business Machines 681 Moore Road King of Prussia, PA 19406
Krell	Apple® II or Apple® II Plus with 48K memory and 16K memory card	Krell Software Corporation 1320 Stony Brook Road Stony Brook, NY 11790
LCSI	Apple® II or Apple® II Plus with 48K memory and 16K memory card	Logo Computer Systems, Inc. 9960 Cote de Liesse Road Lachine, Quebec Canada H8T 1A1
Terrapin	Apple® II or Apple® II Plus with 48K memory and 16K memory card	Terrapin, Inc. 380 Greene Street Cambridge, MA 02139
Texas Instruments	TI-99/4A™ with 32K memory, cassette recorder, or floppy disk	Texas Instruments P.O. Box 53 Lubbock, TX 79408
TRS-80	TRS-80™ Color Computer: Disk version with 32K memory and drive; or Program Pack version with 16K memory	Radio Shack Education Division 400 Atrium 1 Tandy Center Fort Worth, TX 76102

ATARI® 400™ and ATARI® 800™ are registered trademarks of Atari, Inc. (a Warner Communications Company). PET 64™ is a trademark of Commodore Business Machines. Apple® II and Apple® II Plus are registered trademarks of Apple Computer, Inc. TI-99/4A™ is a trademark of Texas Instruments. TRS-80™ is a trademark of Radio Shack, a division of Tandy Corporation.

THE SCHOOL OF THE FUTURE

Papert predicts that the traditional school setting with which we are familiar today will be obsolete before the turn of the century. This may be an unsettling idea to many readers who see themselves educated in conventional environments.

Although Logo can be used easily on home computers, homes of the future should not take over the role of classrooms of today. It is vital that traditional school environments be replaced by settings where children can meet and interact with each other. Papert suggests that "we will at last be able to invent places for children designed for their development as full human beings without having to compromise this noble social purpose by also serving the need to force unwilling children to learn phonics and multiplication tables."[11]

To begin to renovate classrooms as we know them today in order to accommodate them to tomorrow's computer-oriented society, Papert suggests supplying each school child with his or her own personal computer. This may seem unrealistic because of the amount of money required to purchase a vast number of microcomputers to suit this purpose. Papert does not believe this viewpoint to be valid, however.

The current cost of providing a child with thirteen years of formal education is more than $20.000. For children who will be entering the schools within the next five years, the cost could approximate $30,000 per child. Papert estimates that purchasing, updating, and maintaining a personal computer for each school child would be $1,000. The computer would serve the expanding needs of children throughout their careers in school.

The computers would be approximately the size of an average paperback book, asserts Papert. They would be used by either loading cartridges into them or by being connected to a larger computer by a telephone line for information. They would talk, be capable of animation, and allow for interaction.

In order to absorb the expense of providing children with small computers, schools could shorten the conventional thirteen years of schooling to twelve. The impact of computers upon education would be such that students

---

[11]Seymour Papert, "Society Will Balk, But the Future May Demand a Computer for Each Child," *Electronic Education* 1, no. 1 (September 1981): 5-6, 31.

would learn the same amount, or perhaps even more, in shorter periods of time. Computers in the classroom also free teachers to give more individualized attention to each student; therefore, class size could be increased slightly as another way of recuperating the initial cost of the computers.

These small computers would serve numerous functions in the students' lives. They might be used as electronic sketchpads, allowing motion and form. Or, they might serve as writing pads, having the advantage of immediate editing capabilities. Children might use them as electronic games, newspapers, or music synthesizers. The important point is that children would learn to accept computers as friendly, practical, and useful tools.

Some educators may feel that Logo is detrimental to the development of language skills since language and mathematics are often seen as separate, sometimes conflicting skills. Papert points out, however, that Logo can be instrumental in developing well-rounded communication skills. Computers can fill the gap between speaking and writing for two major reasons. First, using a keyboard is a less complex skill than is handwriting, a fact that makes Logo accessible to very young children. Second, a child must learn to write explicit instructions for the computer to follow since speaking is insufficient in the Logo experience.

Clearly, Logo offers a great number of benefits to its users. In order to set up an environment in which Logo can be used to fullest advantage, it is necessary to explore the equipment and software that comprise the Logo experience.

SUMMARY

As Logo systems come into wider use and further research is done to assess their efficacy, more benefits will surface to demonstrate how advantageous Logo can be.

The schools and parents of today are charged with the responsibility of preparing children to become the computer literate adults of tomorrow. Computer scientists now predict that we will use computers in our homes, our workplaces, and in many other facets of our lives. The U.S. Bureau of Labor Statistics foresees a great demand for professionals with computer training within the next decade. It is imperative that children be adequately prepared to face the future and tomorrow's technology. Logo is one effective way to insure that our children will be able to live and work comfortably with computers.

Answers to Practice Activities

The following programs are examples of possible answers for the Practice Activities, for which there are many different correct answers. These programs, which were written by a junior high school student, are intended to serve as a guide. In some cases, the activities suggest exploration by the user; in these cases, an E is placed by the activity number.

Page 8:

1.
```
FD 20
RT 90
FD 20
RT 90
FD 20
RT 90
FD 20
```

2.
```
BK 20
LT 90
BK 20
LT 90
BK 20
LT 90
BK 20
```

3.
```
FD 100
RT 90
FD 10
RT 90
FD 100
RT 90
FD 10
```

4.
```
BK 10
LT 90
FD 100
RT 90
FD 10
LT 90
BK 100
```

Page 9:

1.
```
RT 90
FD 30
RT 90
FD 30
```

LT 90
FD 30
RT 90
FD 30
LT 90
FD 30

2. LT 90
FD 30
LT 90
FD 30
RT 90
FD 30
LT 90
FD 30
RT 90
FD 30

3. FULLSCREEN
RT 90
FD 45
RT 90
FD 45
LT 90
FD 45
RT 90
FD 45
LT 90
FD 45

4. RT 90
FD 10
RT 90
FD 10
LT 90
FD 10
RT 90
FD 10
LT 90
FD 10
RT 90
FD 10

Page 10:

1. FD 40
RT 90
FD 60
RT 90
FD 20
LT 90
FD 20
RT 90

```
FD 20
RT 90
FD 80
HT
```

2.
```
FD 40
RT 45
FD 40
RT 90
FD 40
RT 45
FD 40
RT 90
FD 15
RT 90
FD 10
LT 90
FD 10
LT 90
FD 10
RT 90
FD 30
HT
```

3.
```
LT 90
FD 50
RT 90
FD 10
RT 90
FD 25
LT 90
FD 5
RT 90
FD 17
LT 90
FD 8
RT 90
FD 8
RT 90
FD 8
LT 90
FD 17
RT 90
FD 5
LT 90
FD 25
RT 90
FD 10
RT 90
FD 50
HT
```

Page 12:

1. PU
   FD 100
   RT 90
   FD 50
   PD
   RT 90
   FD 20
   LT 90
   FD 20
   LT 90
   FD 20
   LT 90
   FD 20
2. PU
   FD 100
   LT 90
   FD 50
   PD
   FD 20
   LT 90
   FD 40
   LT 90
   FD 20
   PU
   FD 100
   PD
   FD 20
   LT 90
   FD 40
   LT 90
   FD 20
3. RT 90
   FD 100
   LT 90
   FD 30
   LT 90
   FD 100
   LT 90
   FD 30
   LT 90
   PE
   FD 100
   LT 90
   FD 30
   LT 90
   FD 100
   LT 90

```
FD 30
PD
```

Page 13:

1. E
2. E
3. E
4. E
5. CS
   ```
   PU
   LT 90
   FD 50
   RT 90
   PD
   FD 50
   RT 45
   FD 75
   RT 90
   FD 75
   BK 20
   LT 135
   FD 20
   LT 90
   FD 10
   LT 90
   FD 10
   BK 10
   RT 90
   BK 10
   LT 90
   FD 20
   LT 45
   FD 20
   RT 45
   FD 50
   RT 90
   FD 25
   RT 90
   FD 20
   LT 90
   FD 10
   LT 90
   FD 20
   RT 90
   BK 10
   FD 80
   PU
   RT 90
   ```

```
FD 35
RT 90
FD 20
PD
FD 10
RT 90
FD 10
RT 90
FD 10
RT 90
FD 10
HT
```

6.
```
HT
PU
LT 90
FD 50
RT 90
PD
FD 50
RT 90
FD 100
RT 90
FD 50
BK 50
LT 135
FD 15
LT 135
FD 100
LT 45
FD 15
LT 135
FD 12
RT 90
FD 40
BK 40
LT 90
FD 88
LT 45
FD 15
RT 135
FD 52
```

Page 14:

1. E
2. E

Page 15:

E

Page 19:

1. TO ARROW
   CS
   FD 50
   LT 90
   FD 20
   RT 135
   FD 40
   RT 90
   FD 40
   RT 135
   FD 20
   LT 90
   FD 50
   RT 90
   FD 15
   HT
   END
2. TO TRI
   CS
   RT 180
   PU
   FD 50
   RT 180
   PD
   RT 45
   FD 50
   RT 90
   FD 50
   RT 135
   FD 72
   END

Page 26:

1. CS
   CIRCLER 10
   CIRCLER 20
   CIRCLER 30
   CIRCLER 40
   CIRCLER 50
   HT

or ARCR1 1 36
   ARCR1 10 36
   ARCR1 5 36
   ARCR1 8 36
   ARCR1 3 36

2. CS
   ARCL 20 50
   PU
   HOME
   PD
   ARCR 20 50
   PU
   HOME
   PD
   ARCL 30 80
   PU
   HOME
   PD
   ARCR 30 80
   PU
   HOME
   PD
   ARCL 50 90
   PU
   HOME
   PD
   ARCR 50 90
   PU
   HOME
   PD
   ARCL 70 110
   PU
   HOME
   PD
   ARCR 70 110
   HT
3. PU
   LT 90
   FD 100
   RT 90
   PD
   CIRCLEL 10
   ARCR 50 90
   ARCL1 5 36
   PU
   HOME
   RT 90
   FD 100
   LT 90
   PD
   CIRCLER 10
   ARCL 50 90
   ARCR1 5 36
   HT

Page 30:

1. FULLSCREEN
   REPEAT 4[FD 10 RT 90]
   REPEAT 4[FD 40 RT 90]
   HT
2. FULLSCREEN
   HT
   PU
   LT 90
   FD 100
   RT 90
   PD
   REPEAT 360[FD 2 RT 1]
3. FULLSCREEN
   REPEAT 360[FD 1 RT 1]
   REPEAT 72[RT 5 FD 1]
   REPEAT 36[RT 10 FD 1]
4. FULLSCREEN
   RT 30
   REPEAT 3[FD 80 RT 120]

Page 31:

1. TO SQUARE
   REPEAT 4[FD 50 LT 90]
   END
2. TO SQUARES
   FULLSCREEN
   REPEAT 4[SQUARE LT 90]
   END
3. REPEAT 10[CIRCLER 10 PU FD 1000 RT 190 PD]

Page 33:

1. TO TRIANGLE
   FULLSCREEN
   RT 30
   FD 50
   RT 120
   FD 50
   RT 120
   FD 50
   END

   TO TRIANGLES
   REPEAT 4[TRIANGLE PU FD 20 PD]
   END
2. TO CIRCULAR
   CS

```
REPEAT 36[FD 60 BK 60 RT 10]
END
```

3.
```
TO HORIZ
CS
FULLSCREEN
PU
BK 100
RT 90
BK 100
PD
REPEAT 9[FD 200 BK 200 LT 90 PU FD 25 RT 90 PD]
END
```

Page 40:

1. Enter ERALL. Enter procedures for TABLE and CHAIRS from pp. 62–63.

```
TO LAMP
HT
PU
HOME
FD 60
RT 90
FD 50
PD
LT 90
ARCR1 5 9
CIRCLER 5
CIRCLER 15
LT 90
FD 20
BK 20
RT 90
ARCR1 5 9
END
SAVE "ROOM
```

Turn computer off, load Logo, and enter:

```
LOAD "ROOM
POTS
```

2.
```
TO TREE
FULLSCREEN
SETPC 2
PU
HOME
BK 75
RT 90
FD 10
PD
LT 90
```

```
BK 30
FD 30
ARCR1 10 5
PU
HOME
BK 75
LT 90
FD 10
PD
RT 90
BK 30
FD 30
ARCL1 10 5
PU
HOME
BK 40
RT 90
PD
CIRCLEL 50
HT
END

TO FLOWER
FULLSCREEN
PU
HOME
RT 90
FD 100
RT 90
FD 50
PD
SETPC 5
REPEAT 5[ARCR1 2 10 RT 90 ARCR1 2 10]
END

PACKAGE "SCENE [TREE FLOWER]
SAVE "SCENE "SCENE
```

3.
```
LOAD "ROOM
LOAD "SCENE
PO [TABLE CHAIR LAMP]
POPS "SCENE
```

4.
```
ERALL "SCENE
POTS "SCENE
```

5.
```
ERALL
POTS
```

Page 46:

1.
```
E
PACKAGE "ONE "ONE
SAVE "ONE "ONE
```

2. PACKAGE "I "I
   SAVE "I "I
3. PACKAGE "ONEI [I ONE]
   SAVE "ONEI "ONEI
4. Edit I by changing the name to BLUEI and adding SETPC 5 at line 2.

   PACKAGE "BLUEI "BLUEI
   SAVE "BLUEI "BLUEI
5. E

Page 50:

1. TO CIRCLES
   CS
   PD
   CIRCLEL 40
   PE
   CIRCLEL 40
   PD
   CIRCLER 40
   PE
   CIRCLER 40
   CIRCLES
   END
2. TO VLINES
   SETBG 5
   SETPC 1
   HT
   PD
   FD 100
   BK 200
   PE
   FD 200
   RT 90
   PU
   FD 20
   LT 90
   BK 100
   VLINES
   END
3. TO BOXES
   HT
   PD
   REPEAT 4[FD 15 RT 90]
   PU
   CLEAN
   RT 45

```
FD 25
BOXES
END
```

Page 53:

1.
```
PU
RT 45
FD 150
RT 180
PD
FD 500
```
2.
```
TO CIRCS
WRAP
PD
CIRCLER 40
PU
RT 90
FD 50
LT 90
PD
CIRCS
END
```

In this program replace WRAP with FENCE, then with WINDOW.

Page 54:

1. 120
2. 140
3. -140
4. -120

Page 55:

1. DOT [0 118]
2. DOT [70 60]
3. DOT [-70 -60]
4. FULLSCREEN
   DOT [-135 -110]
5. DOT [130 118]
6. E

Page 56:

1.
```
PD
FULLSCREEN
HT
DOT [100 100]
```

```
DOT [-76 -93]
DOT [-98 78]
DOT [56 -36]
PU
SETPOS [100 100]
PD
SETPOS [-76 -93]
SETPOS [-98 78]
SETPOS [56 -36]
SETPOS [100 100]
```

2.
```
TO STARSIX
FULLSCREEN
PD
DOT [100 -12]
DOT [-100 -12]
DOT [0 88]
DOT [100 38]
DOT [-100 38]
DOT [-50 -38]
PU
SETPOS [100 -12]
PD
SETPOS [-100 -12]
SETPOS [0 88]
SETPOS [100 -12]
PU
SETPOS [100 38]
PD
SETPOS [-100 38]
SETPOS [0 -38]
SETPOS [100 38]
END
```

3.
```
CS
PU
HT
SETPOS [-50 -75]
PD
SETPOS [-75 -75]
SETPOS [-75 -50]
SETPOS [-50 -50]
SETPOS [-50 -75]
```

4.
```
TO BORDER
FULLSCREEN
PU
CS
HT
SETPOS [-138 -118]
PD
```

```
SETPOS [-138 118]
SETPOS [138 118]
SETPOS [138 -118]
SETPOS [-138 -118]
SETPOS [0 -118]
SETPOS [0 118]
PU
SETPOS [-138 0]
PD
SETPOS [138 0]
END
```

Page 57 (first set):

1.
```
TO HLINES
HT
PU
SETX -120
PD
SETX 120
PE
SETX -120
FD 10
HLINES
END
```

2.
```
TO VTLINES
HT
PU
SETY 115
PD
SETY -115
PE
SETY 115
PU
RT 90
FD 10
LT 90
VTLINES
END
```

Page 57 (second set):

```
FD 50
SETH 90
FD 50
SETH 180
FD 50
SETH 270
FD 50
SETH 45
```

```
FD 71
SETH 0
BK 50
SETH 315
FD 72
```

Page 58:

1. HT
   ```
   SETH TOWARDS [100 0]
   FD 100
   SETH TOWARDS [100 100]
   FD 100
   SETH TOWARDS [0 100]
   FD 100
   SETH TOWARDS [0 0]
   FD 100
   ```
2. Repeat No. 1 above and add these lines:
   ```
   SETH TOWARDS [100 100]
   FD 140
   SETX 0
   SETH TOWARDS [100 0]
   PD
   FD 140
   PU
   SETY 50
   SETH TOWARDS [0 50]
   PD
   FD 100
   PU
   SETY 100
   SETX 50
   PD
   SETH TOWARDS [50 0]
   FD 100
   ```
3. TO OUTLINE1
   ```
   FULLSCREEN
   HT
   PU
   SETPOS [-138 -118]
   PD
   SETPOS [-138 118]
   SETPOS [138 118]
   SETPOS [138 -118]
   SETPOS [-138 -118]
   END

   TO OUTLINE2
   PU
   ```

```
SETPOS [-120 -100]
PD
SETPOS [-120 100]
SETPOS [120 100]
SETPOS [120 -100]
SETPOS [-120 -100]
END

TO OUTLINE3
PU
SETPOS [-100 -80]
PD
SETPOS [-100 80]
SETPOS [100 80]
SETPOS [100 -80]
SETPOS [-100 -80]
END

TO OUTLINES
OUTLINE1
OUTLINE2
OUTLINE3
END
```

Page 61:

1.
```
ED "PLAYGROUND
SETBG 1
SETPC 0
CHAINLINK
SETPC 5
BALLOONS
END

ERASEFILE "PLAYGROUND
SAVE "PLAYGROUND
```

2.
```
TO KITE
SETPC 4
HT
PU
HOME
FD 90
LT 90
FD 100
PD
SETH 45
REPEAT 4[FD 20 LT 90]
END
```

3. TO STRING
   SETPC 3
   PU
   HOME
   HT
   FD 90
   LT 90
   FD 100
   PD
   SETH 200
   FD 70
   END
4. TO LIGHTPOST
   PU
   HOME
   HT
   SETPC 2
   SETPOS [-124 24]
   PD
   SETH 0
   FD 30
   SETH 270
   CIRCLER 6
   END
5. ED "PLAYGROUND
   TO PLAYGROUND
   SETBG 0
   SETPC 1
   CHAINLINK
   FD 139
   LT 90
   SETPC 5
   BALLOONS
   KITE
   STRING
   LIGHTPOST
   HT
   END

   ERASEFILE "PLAYGROUND
   SAVE "PLAYGROUND

Page 63 (first set):

Edit CHAIRS procedure to include these lines:

SETBG 3	(at the beginning of the program)
SETPC 1	(after the first PD)
SETPC 5	(after the second PD)

Page 63 (second set):

1. See p. 13, Activity No. 6.
2. E
3. E

Page 64 (first set):

1. Edit PANE and call it LPANE by entering the following seven lines and omitting the original first seven lines.

```
TO LPANE
PU
HOME
FD 75
LT 90
PD
FD 50
```

2. Edit PANE to add SETPC 4 at line 16 (just before RT 90).

Page 64 (second set):

1.
```
TO ROOM
CHAIRS
TABLE
PANE
END

PACKAGE "ROOM[CHAIRS TABLE PANE ROOM]
```

2. SAVE "ROOM "ROOM
3. E
4. ERALL

Page 65 (first set):

1.
```
TO BIRDS
PU
HOME
HT
SETPC 1
SETH 315
FD 130
PD
BK 20
SETH 45
FD 20
SETH 90
PU
FD 150
```

```
RT 90
FD 30
SETH 300
PD
FD 20
BK 20
SETH 60
FD 20
PU
SETH 90
BK 90
SETH 100
PD
FD 20
BK 20
SETH 260
FD 20
END
```

2.
```
TO SHORE
SETPC 5
PU
HT
FULLSCREEN
HOME
PD
LT 90
FD 277
PU
LT 90
FD 100
RT 90
FD 100
SETH 60
PD
REPEAT 23[FD 10 RT 2]
END
```

3.
```
TO BOAT
SETH 340
FD 30
SETH 200
FD 30
SETH 145
FD 9
BK 9
SETH 90
FD 20
SETH 215
FD 9
```

```
SETH 270
FD 7
END

TO BOATS
FULLSCREEN
HT
PU
HOME
SETPC 2
RT 180
FD 95
LT 90
PD
BOAT
PU
BK 100
RT 90
FD 30
PD
BOAT
END
```

4.
```
TO SEASCAPE
BIRDS
SHORE
BOATS
END

PACKAGE "SEASCAPE[BIRDS SHORE BOAT BOATS
    SEASCAPE]
SAVE "SEASHORE "SEASHORE
```

Page 65 (second set):

1.
```
TO ROAD
SETPC 1
PU
FULLSCREEN
HT
SETPOS[81 -110]
SETH 30
PD
FD 110
PU
SETPOS[-21 -110]
SETH 60
PD
FD 100
```

```
PU
FD 38
PD
FD 37
END
```

2.
```
TO CABIN
SETPC 5
PU
SETPOS[98 -24]
PD
SETH 180
FD 25
RT 90
FD 10
RT 90
FD 10
LT 90
FD 5
LT 90
FD 10
RT 90
BK 10
FD 36
RT 90
FD 25
RT 90
FD 42
PU
SETPOS[57 -24]
SETH 45
PD
FD 30
SETH 135
FD 30
BK 5
SETH 0
FD 10
LT 90
FD 5
LT 90
FD 4
PU
SETPOS[95 0]
SETPC 1
PD
CIRCLER 3
PU
SETPOS[100 10]
PD
```

```
CIRCLER 3
PU
SETPOS[96 20]
PD
CIRCLER 3
PU
HT
END
```

3.
```
TO MTS
SETPC 4
PU
SETPOS[125 25]
SETH 315
PD
FD 100
SETH 240
FD 35
SETH 310
FD 50
SETH 200
FD 50
SETH 315
FD 70
SETH 220
FD 85
PU
SETH 270
FD 10
SETH 0
FD 50
SETH 225
PD
FD 57
HT
END
```

4.
```
TO TREESM
PD
SETH 0
FD 10
RT 90
FD 5
SETH 340
FD 15
SETH 200
FD 15
SETH 90
FD 5
PU
END
```

```
TO TREELG
PD
SETH 0
FD 15
RT 90
FD 10
SETH 340
FD 30
SETH 200
FD 30
SETH 90
FD 10
PU
END

TO TREES
SETPC 2
PU
HT
SETPOS[-100 30]
TREESM
SETPOS[-50 20]
TREESM
SETPOS[0 40]
TREESM
SETPOS[-80 -40]
TREESM
SETPOS[-110 -60]
TREESM
SETPOS[30 -65]
TREELG
SETPOS[120 -110]
TREELG
END

TO COUNTRYROAD
ROAD
CABIN
MTS
TREES
END
```

5. Colors have been inserted into the above programs.

6.
```
PACKAGE "COUNTRYROAD[ROAD CABIN MTS TREESM
    TREELG TREES COUNTRYROAD]
SAVE "COUNTRYROAD "COUNTRYROAD
```

Page 68:

1. TO LINE1 :LINE
   CS
   FD :LINE
   END

2a. STAIRS :STEPSIZE

2b. Twenty (20) will be substituted for STEPSIZE in the REPEAT line.

2c. 12

2d. E

3. TO ZIGZAG :LINE
   PU
   HOME
   CS
   SETPOS[-120 0]
   RT 45
   PD
   FENCE
   REPEAT 20[FD :LINE RT 90 FD :LINE LT 90]
   END

Page 70:

1. TO ZIG2 :LINE :ANGLE
   FULLSCREEN
   PU
   HOME
   WRAP
   CS
   RT 90
   FD 120
   LT 135
   PD
   REPEAT 50[FD :LINE LT :ANGLE FD :LINE RT :ANGLE]
   END

   Try entering:

   ZIG2 20 60
   ZIG2 15 100
   ZIG2 25 125

2. TO STAIRS2 :VLINE :HLINE
   FULLSCREEN
   HT
   PU
   SETPOS[90 -100]
   PD
   REPEAT 20[FD :VLINE LT 90 FD :HLINE RT 90]
   END

Try entering:

```
STAIRS2 10 50
STAIRS2 40 25
```

3.
```
TO DESIGN3 :LINE1 :RTURN :LINE2
FULLSCREEN
CS
HT
FD :LINE1
RT :RTURN
FD :LINE2
RT :RTURN
BK :LINE1
RT :RTURN
BK :LINE2
END
```

Try entering:

```
DESIGN3 20 65 55
DESIGN3 55 30 70
```

Page 71:

1.
```
TO BOX1 :SIDE
REPEAT 4[FD :SIDE LT 90]
END

TO BOXES1 :SIDE
FULLSCREEN
HT
BOX1 :SIDE
PU
FD 50
RT 45
FD 20
PD
BOXES1 :SIDE
END
```

2. E

Page 72:

1.
```
RT 90
LT 180
LT 270
```

or
```
LT 50
RT 130
RT 280
```

2.
```
TO TURN
PU
HOME
SPLITSCREEN
```

```
REPEAT 8[LT 45 PR HEADING]
END
```

Page 73:

1.
```
PD
CS
HOME
FD 100
PR POS

CS
SETX 100
SETY 0
PR POS

CS
FULLSCREEN
RT 90
FD 60
RT 90
FD 110
PR POS
```
(Press CTRL S to see answer.)
```
CS
LT 90
FD 50
RT 90
FD 50
PR POS

CS
BK 40
LT 90
FD 100
PR POS
```

2.
```
TO LOCATE
CS
SPLITSCREEN
BK 50
RT 35
PR POS
FD 80
LT 60
PR POS
RT 10
FD 100
PR POS
RT 90
BK 125
PR POS
END
```

Page 74:

1a. CS
RT 90
FD 20
PR XCOR

1b. CS
RT 90
FD 75
PR XCOR

1c. CS
RT 90
BK 115
PR XCOR

1d. CS
RT 90
BK 120
PR XCOR

2a. CS
FD 110
PR YCOR

2b. CS
RT 180
BK 55
PR YCOR

2c. CS
BK 90
PR YCOR

2d. CS
RT 180
FD 10
PR YCOR

3. E

Page 77:

1. CS
LT 45
BK 50
RT 75
FD 150
MAKE "POSNOW POS
LT 120
FD 50
LT 60
FD 180
RT 90
FD 45
SETPOS :POSNOW

2.
```
CS
LT 50
FD 70
RT 100
BK 105
MAKE "XNOW XCOR
PR XCOR
MAKE "YNOW YCOR
PR YCOR
RT 45
FD 60
RT 140
BK 200
SETX :XNOW
SETY :YNOW
PR XCOR
PR YCOR
```

3.
```
TO HOUSE3
PU
CS
LT 90
FD 30
RT 90
PD
MAKE "START POS
FD 50
MAKE "CORNER1 POS
LT 45
FD 50
LT 90
FD 50
MAKE "CORNER2 POS
SETPOS :CORNER1
SETPOS :CORNER2
SETH 180
FD 50
SETPOS :START
HT
END
```

Page 79:

1. 889
   7877
   8578
   9.78
   187.81
2. 42
   3027

32190
-902.73
428.1

3. 30
   2992
   2112
   2325
   15975
   37672.
   90.88

4. 10.
   8.66666
   15.6818
   100.589
   .266667
   556.72

Page 83:

1.
```
TO RECTANGLE
REPEAT 2[FD 20 RT 90 FD 40 RT 90]
END

TO RECTANGLES1 :ANGLE
RECTANGLE
RT :ANGLE
RECTANGLES1 :ANGLE + 10
END
```

2.
```
TO STARS :SEGMENT
PU LT 90
FD :SEGMENT
PD
STAR :SEGMENT
STARS :SEGMENT *2
END
```

3.
```
TO STARS :SEGMENT
CS
PU LT 90
FD :SEGMENT - 5
PD
STAR :SEGMENT
STARS :SEGMENT / 2
END
```

Page 86:

1. Change the REPEAT line in PEAKS:
   REPEAT 50[RT 90 FD RANDOM 25 LT 90 FD RANDOM 25]

2. TO RANCIRS
   PU FULLSCREEN WRAP
   SETX RANDOM 120
   SETY RANDOM 120
   PD
   CIRCLEL RANDOM 50
   RANCIRS
   END
3. TO RANSQS :SIDE
   PU FULLSCREEN
   SETX RANDOM 120
   SETY RANDOM 120
   PD
   REPEAT 4[FD :SIDE RT 90]
   RANSQS :SIDE
   END
4. Add SETPC RANDOM 6 to line 2 of the DOTS procedure from p. 84.

Page 87:

1. 88.3636
   89.1637
   89.3635
2. -1
   0
   .707107
   -.866025
3. 34
   20
   99
   478
4. 4
   3
   23
   496
5. 436
   85
   35
   100
6. .587785
   .866025
   .987688
   .5
7. 6.
   18.9737
   31.607
   31.6386

Page 92:

1.
```
TO CNT1 :NUMBER
IF :NUMBER > 20[STOP]
PR :NUMBER
CNT1 :NUMBER + 1
END
```

```
TO CNT1 :NUMBER
PR :NUMBER
IF :NUMBER = 20[STOP]
CNT1 :NUMBER + 1
END
```

In the second example, if the NUMBER is greater than 20, the counting will begin with the NUMBER input and go on continuously.

2a.
```
TO CNTBK1 :NUMBER
IF :NUMBER = 0[STOP]
PR :NUMBER
CNTBK1 :NUMBER - 1
END
```

2b.
```
IF 0 = :NUMBER[STOP]
```

2c.
```
IF :NUMBER < 1[STOP]
```

Page 93:

1.
```
TO COUNT100 :NUMBER
IF OR :NUMBER < 100 :NUMBER > 200[STOP]
PR :NUMBER
COUNT100 :NUMBER + 1
END
```

2.
```
TO COUNT5 :NUMBER
IF OR :NUMBER < 1 :NUMBER > 100[STOP]
PR :NUMBER
COUNT5 :NUMBER + 5
END
```

3.
```
TO CTBK :NUMBER
IF OR :NUMBER > 50 :NUMBER < 10[STOP]
PR :NUMBER
CTBK :NUMBER - 1
END
```

Enter:

```
CTBK 50
```

4.
```
TO CBK :NUMBER
IF OR :NUMBER > 200 :NUMBER < 150[STOP]
PR :NUMBER
```

```
CBK :NUMBER - 2
END
```

Enter:

```
CBK 200
```

Page 94:

1. Replace the IF line in COUNT with the following:

```
IF AND :NUMBER > 99 :NUMBER < 200[STOP]
```

2.

```
TO C1 :NUMBER
IF AND :NUMBER > 10 :NUMBER < 20[STOP]
PR :NUMBER
IF :NUMBER = 50[STOP]
C1 :NUMBER + 1
END
```

Page 96:

1. Omit SETPOS [-140 60]
   Change RT 20 to LT 20
2. E

Page 98:

1. Add these lines after BUBBLES in UNDERWATER:

```
FULLSCREEN
SETPC 5
PU
SETPOS[-130 80]
SETH 45 PD
REPEAT 45[FD 4 RT 90 FD 4 LT 90]
PLANT
HT
END
```

```
TO PLANT
PU SETPC 2
SETPOS[70 -100]
SETH 45 PD
REPEAT 10[FD 4 LT 90 FD 4 RT 90]
PU SETPOS[70 -100]
PD
SETH 45
ARCR1 5 5 PU
SETPOS[70 -100]
SETH 315 PD
ARCL1 5 5 PU
SETPOS[70 -100]
```

```
SETH 20 PD
ARCR1 5 10 PU
SETPOS[70 -100]
SETH 340 PD
ARCL1 5 10
END
```

2. Add this line to UNDERWATER after PLANT:

```
SMFISH
```

```
TO SMFISH
PU HT SETPOS[70 20]
PD LT 45
ARCL 20 150
PU SETPOS[70 20]
PD RT 45
ARCR 20 150
DOT[64 20]
DOT[74 20]
DOT[76 24]
DOT[74 28]
DOT[76 32]
DOT[74 36]
DOT[76 40]
END
```

Page 99:

1. Edit FOOD:

```
TO FOOD
REPEAT 9[FOLIAGE]
END

TO FOLIAGE
SETPC 2
SETH 315
REPEAT 4[FD 3 LT 90 FD 3 RT 90]
LT 45
PU FD 15 PD
SETH 315
END
```

2. Run this procedure before running HUNGRYWORM:

```
TO SUN
SETPC 4
HT PU
SETPOS [80 80]
PD CIRCLER 20
PU HOME PD
END
```

3a. TO PAWS
```
HT PU
SETPOS[50 -50] PD
DOT[50 -50] DOT[54 -46] DOT[50 -46]
DOT[42 -42] DOT[46 -38] DOT[42 -38]
DOT[34 -34] DOT[38 -30] DOT[34 -30]
```
Note: Continue with DOT commands decreasing the coordinate values until 0 is reached and then adjust the values accordingly.

3b. TO FILLUP
```
HT PU WRAP FULLSCREEN
SETPOS[-138 -115]
SETPC 5
PD
SETH 89.95
REPEAT 25[FD 10000]
END
```

Page 105:

1. TO INITIALS
```
PR FIRST "MATTHEW
PR FIRST "ERNEST
PR FIRST "WATSON
END
```
2. TO CHOOSE
```
PR[PICK A NUMBER BETWEEN 1 and 6]
MAKE "NUMBER READCHAR
IF :NUMBER < 4[PR[ONE, TWO, THREE, SWEET AS A
  BEE]STOP]
IF :NUMBER > 3[PR[THREE, FOUR, FIVE, THEY'RE IN
  THE HIVE.]STOP]
END
```
3. TO SINGULAR
```
PR BUTLAST "PICNICS
PR BL "WATERMELONS
PR BL "ANTS
END
```
4. E
5. E

Page 109:

1. E
2. E
3. E

Page 114:

1. 240
2. 280
3. 350
4. It depends on your monitor.
5. Tall and skinny:

```
FD 100
RT 90
FD 20
RT 90
FD 100
RT 90
FD 20
```

Short and fat:

```
FD 20
RT 90
FD 100
RT 90
FD 20
RT 90
FD 100
```

Outlining the screen:

```
TO BORDER
FD 118
RT 90
FD 138
RT 90
FD 235
RT 90
FD 275
RT 90
FD 235
RT 90
FD 138
END
```

6. Large triangle:

```
FD 100
RT 120
FD 100
RT 120
FD 100
```

Small triangle:

```
FD 20
RT 120
FD 20
```

```
RT 120
FD 20
```

All sides are equal in both triangles above.

7. Letter N:

```
FD 100
RT 140
FD 130
LT 140
FD 100
```

Letter T:

```
FD 100
RT 90
BK 50
FD 100
```

Letter H:

```
FD 100
BK 50
RT 90
FD 50
RT 90
BK 50
FD 100
```

Page 117:

1. Using PE:

```
REPEAT 4[FD 50 RT 90]
PE
REPEAT 4[FD 50 RT 90]
```

Using PX:

```
PD
REPEAT 4[FD 50 RT 90]
PX
REPEAT 4[FD 50 RT 90]
```

2.
```
PU
FD 25
PD
REPEAT 4[FD 25 RT 90]
PU
BK 25
```

3.
```
CS
REPEAT 2[FD 50 RT 90 FD 20 RT 90]
FD 25
RT 90
BK 10
PE
FD 50
```

```
HOME
PD
RT 90
FD 10
LT 90
PE
BK 10
FD 150
PD
```

4.
```
REPEAT 2[FD 40 RT 90 FD 100 RT 90]
FD 40
RT 60
FD 59
RT 60
FD 59
```

Page 119:

1.
```
CS
FD 100
RT 90
FD 60
HOME

CS
FD 50
LT 45
FD 80
HOME
```

2.
```
PU
FD 120
RT 90
FD 138
PD
HOME
PU
FD 120
LT 90
FD 138
PD
HOME
PU
FULLSCREEN
BK 118
RT 90
FD 138
PD
```

```
HOME
PU
BK 118
LT 90
FD 138
PD
HOME
```

Page 124:

1. See TABLE procedure on p. 20.
2. See CHAIR procedure on pp. 20–21.
3. See PANE procedure on p. 21.

Page 125:

1.
```
TO RECTANGLE
REPEAT 2[FD 20 RT 90 FD 50 RT 90]
RT 20
REPEAT 18[RECTANGLE]
END
```
2. See ROOM procedure on p. 21.
3.
```
TO REDO
HT
SETPC 2
REPEAT 4[FD 50 RT 90]
RT 3
REDO
END
```

Page 126:

E

Page 133:

1.
```
SETPC 1
CIRCLER 60
SETPC 2
CIRCLEL 50
SETPC 3
CIRCLER 40
SETPC 4
CIRCLEL 30
SETPC 5
CIRCLER 20
SETPC 1
CIRCLEL 10
```

2. CIRCLER 60
   PU
   RT 90
   FD 10
   LT 90
   PD
   CIRCLER 50
   PU
   RT 90
   FD 20
   LT 90 PD
   CIRCLER 30
   PU HOME PD
   CIRCLEL 60
   PU
   LT 90
   FD 10
   RT 90
   PD
   CIRCLEL 50
   PU
   LT 90
   FD 20
   RT 90
   PD
   CIRCLEL 30
3. CIRCLER 60
   CIRCLEL 60
   ARCR 150 50
   PU HOME PD
   ARCL 150 50
   PU HOME PD
   ARCR1 15 10
   PU HOME PD
   ARCL1 15 10
   PU HOME PD
   CIRCLER 20
   CIRCLEL 20

APPENDIX

Suggested Activities

1. Draw a glass and simulate it filling with milk.
2. Simulate a flower growing, adding leaves and flowers.
3. Simulate a tree growing, adding branches and leaves. Use appropriate colors.
4. Draw a small cartoon cat. Make it look like it is winking one eye by using a DOT set to go on and off.
5. Draw a window with an object in it. Simulate a shade coming down, causing the object to disappear.
6. Design a stack of blocks, each one smaller than the one below it. Make each block a different solid color.
7. Develop a sequence of design patterns. Have observers predict what design will be shown next.
8. Design animation to show the four seasons.
9. Create a design which includes several geometric shapes, sizes, and colors. Have observers see how many they can identify.
10. Develop a display to show the Roman numerals to 50 (L).
11. Draw a sun that is solid orange. **Hint:** You will not be able to use CIRCLER or CIRCLEL for this.
12. Write a program to show pictorially how to divide squares, rectangles, hexagons, and circles into halves and sixths.
13. Create an animation to make all the shapes from three sides to ten sides. Have all sides equal.
14. Make a scene with objects filled in with different solid colors.
15. Make an animation of a river flowing and simulate a boat moving down the river.
16. Draw a floor plan to the scale of your home. Include doorways.

17. Draw the four sides and top and bottom views of a cube and a pyramid.

18. Draw the side and top views of your school or house.

19. Create an animated holiday scene.

20. Write a short story and illustrate it using different scenes. Time the appearance of scenes on the screen so that they change automatically at about the time you finish the narration describing them.

21. Design a demonstration program which includes examples of the many different capabilities of Logo. Incorporate everything you have learned.

GLOSSARY/INDEX

Animation, 95–99
ANSWER1 procedure, 108; ANSWER2 procedure, 108; ANSWER3 procedure, 108–109
ARCLEFT (ARCL): Stored procedure, requiring two inputs, that produces an arc to the left, 23–24, 131–132; practice activities, 26, 133
ARCL1: Stored procedure, requiring two inputs, that produces arcs or circles to the left, 25–26, 131–133; practice activities, 26, 133
ARCRIGHT (ARCR): Stored procedure, requiring two inputs, that produces an arc to the right, 23–24, 131–132; practice activities, 26, 133
ARCR1: Stored procedure, requiring two inputs, that produces arcs or circles to the right, 25–26, 131–133; practice activities, 26, 133
ARCTAN: Arithmetic operation that outputs the arctangent of the input value, 86; practice activities, 87
Arithmetic operations, 79–88; in procedures, 80–83, 87–94, inside back cover; practice activities, 79–80, 83, 87–88, 92–94

BACK (BK): Command that indicates the distance the user wishes to move the turtle back, 4, 8, 113; practice activities, 8, 9, 114
BALLOON procedure, 31–32
BALLOONS procedure, 32
BF. *See* BUTFIRST
BK. *See* BACK
BL. *See* BUTLAST
Black and white monitor, 15, 121
Blank disks: Used to save procedures, 34, 126–127
BOX procedure, 27
Brackets, 22–23, 27–28, 37, 54, 87, 114
BUBBLE procedure, 96
BUBBLES procedure, 96–97
Bug: Error the user makes, 6, 123
BUTFIRST (BF): Operation that outputs all but the first element of a word or list, 103; practice activities, 105
BUTLAST (BL): Operation that outputs all but the last element of a word or list, 103; practice activities, 105
BUTTERFLY procedure, 19

Catalog: Listing of all files on a disk, 34–35, 38–39, 126, 128
CHAINLINK procedure, 59–61
CHAIR procedure, 20
CHAIRS procedure, 62
CIRCLEL: Stored procedure, requiring an input, that draws a circle to the left, 24–25, 131–133; practice activities, 26, 133
CIRCLER: Stored procedure, requiring an input, that draws a circle to the right, 24–25, 32, 45, 131–133: practice activities, 26, 133
CLEAN: Command that completely erases the screen, but which does not cause the turtle to return home, 48–50; practice activities, 50–51
CLEARSCREEN (CS): Command that erases all graphics on the screen and places the turtle in home position, 4, 6, 7, 13, 19, 113, 119, 123
CLEARTEXT: Command that erases all text on the screen, 17
Color codes, 13, 120; practice activities, 14
Color control commands, 14, 120

COLORLINE procedure, 89–90
Combining procedures, 59–65
Conditional: Capability of the computer to follow a different instruction when a variable meets specified requirements, 89–94; practice activities, 92, 93, 94; uses of, 89–94
Control characters, 41–43, 117, inside back cover; practice activities, 43–44
Coordinates, 53–58; practice activities, 54–56
COS: Arithmetic operation producing the cosine of the value input, 86; practice activities, 87
COUNT: Operation that determines the number of elements that make up a list, 104
COUNT procedure, 94
COUNT10 procedure, 90; COUNT 100 procedure, 93
COUNTRYROAD activity, 65
CS. *See* CLEARSCREEN
CRAZY procedure, 69
CTRL A: Places the cursor on the first letter of the line without erasing, 5, 6, 41, 125
CTRL B: Moves the cursor back one character at a time without erasing, 41–42, 117, 125
CTRL C: Writes changes into memory and terminates the edit mode, 42, 125
CTRL D: Deletes the character under the cursor, 42, 117, 125
CTRL E: Moves the cursor to the end of the line, 42, 125
CTRL F: Moves the cursor forward one position on the line without erasing, 42, 117, 125
CTRL G: Terminates the edit mode without writing changes into memory when in the edit mode, 42; stops the procedure from running when in the turtle mode, 49, 120, 125
CTRL K: Kills or deletes all characters from the cursor's position to the end of the line, 42
CTRL L (FULLSCREEN): Makes the entire screen available for graphics, 16, 42, 54, 118–119
CTRL N: Moves the cursor to the next line, 42, 125
CTRL O: Opens up a blank line just above the line the cursor is on, 42, 125
CTRL P: Moves the cursor to the previous line, 42, 125
CTRL Q: Alerts the computer that characters are being used to form words by producing a slash (\), 100
CTRL S (SPLITSCREEN): Allots the top twenty lines of the screen to graphics and the bottom four lines to text, 16, 17, 109, 118–119
CTRL T (TEXTSCREEN): Allots the entire screen to text, 16, 17, 22, 32, 118–119
CTRL V: Causes the next "page" of text to appear, 43
CTRL W (WAIT): Temporarily stops whatever is happening, 22, 109, 120; used to stop a recursive command, 49; and arithmetic, 80
CTRL Y: Reinserts a killed line into the position where the cursor is located when in the edit mode, 43; produces the most recent line of instructions on the screen when in the turtle mode, 6
CTRL Z: Used to stop a procedure during execution, 49
Cursor: Flashing white box that indicates user's place in a line of instructions, 3, 5, 7, 112, 117

Deleting procedures from disk. *See* ERASEFILE
DIAMOND procedure, 30
Disks: Used to save procedures, 34, 126–127
Distortion on monitor, 10
DOT: Command that places a dot anywhere on the screen, given the coordinates *x* and *y*, 54, 55; practice activities, 55
DOTS procedure, 84
Dynaturtles, 141

EDIT (ED): Used with a procedure name to enter the edit mode, 41, 125
Editing, control characters for, 41–43, 123; of saved procedures, 44; of saved procedures (saved more than once), 45–46; practice activities, 43–47, 126
Editor: Mode for writing and making changes in procedures, 41–47
Efficiency: Ability to write desired programs in fewest number of lines possible, 62, 66
Elements: Characters, numbers, or letters that make up a word, 100
END: Command that signals the completion of a procedure, 18, 32, 123
ERASE (ER): Command used to erase procedures the user does not want, 37–38, 129; practice activities, 40
ERASEALL (ERALL): Command used for erasing everything in memory or entire packages of procedures, 38, 41, 131
ERASEFILE: Command that deletes files from the disk, 45–46, 129–131
Error correction, 4, 6, 112, 123
Error messages, 6, 16, 52, 66, 89, 101, 118, inside back cover
EYE procedure, 97

FD. *See* FORWARD
FENCE: Command that instructs the turtle not to go outside of the screen boundaries, 52, 86; practice activities, 53
FIRST: Operation that outputs only the first element of a word or list, 103; practice activities, 105
FISH procedure, 96–97
Floor turtle, 140–141
FOOD procedure, 98–99
Formatting: Initializing a blank disk, 34–35, 126–127
FORWARD (FD): Command that indicates the distance the user wishes the turtle to travel on the screen in whatever direction the turtle is pointed, 3, 8, 112–113; practice activities, 8, 9, 114
FULLSCREEN (CTRL L): Makes the entire screen available for graphics, 16, 42, 54, 118–119

Grid of *x* and *y* coordinates, 53

HEAD procedure, 98–99
Heading: Direction which the turtle is facing, 4
HEADING: Operation used to determine the current direction the turtle is facing, 72; practice activities, 72
HIDETURTLE (HT): Command that causes the turtle to disappear from the screen, 13, 54, 115; practice activities, 13
HOME command, 17, 22, 48, 50, 119, 132
Home position: The turtle is located in the center of the screen, facing or heading straight up, 3, 7, 13, 115

HORIZON procedure, 86
HT. *See* HIDETURTLE
HUNGRYWORM procedure, 98–99

IF: Command or operation that initiates a conditional (*see* Conditional); practice activities, 92–94; with AND, 94; with OR, 92–93
Initializing: Preparing a blank disk for loading and saving procedures, 34–35, 126–127
Input: A value that is entered along with a command, 66, 90; internally generated, 74–76; multiple, 68–70; practice activities, 68, 70, 71; used in procedures within procedures, 70–71
INT: Arithmetic operation that deletes any numbers following the decimal point in a value input, 87; practice activities, 87

Lamplighter School, 142–143
LAST: Operation that prints only the last element of a word or list, 103; practice activities, 105
LEFT (LT): Command that turns the turtle to the left the number of degrees indicated, 4, 8, 113; practice activities, 8, 9, 114
Left arrow, 5, 42
Library procedures, 23–26
Limits. *See* Screen limits
Light turtle, 140–141
List, 100
LOAD: Command that places named file into computer's memory or workspace, 38–39, 128
Loading procedures, 39
Logo, accessibility to children, 137; and social skills, 147–148; as an educational tool, 146–147; at Lamplighter School, 142–143; epistemological benefits of, 145; hardware required, 152; history, 139; in adult education, 150; in art, 147; in schools of the future, 154–155; in special education, 149–150; learning the language of, 139; loading into the computer, 3; modes of, 152; objections to, 145–146; origin of name, 139; prerequisites for learning, 140; software required, 2; steps for use, 112; theoretical underpinnings of, 144–145
LOGO EDITOR: These words appear across the bottom of the screen to indicate that the user is in the edit mode, 41
LT. *See* LEFT

MAGICSQUARE1 procedure, 48; MAGICSQUARE2 procedure, 49
MAGICSTAR procedure, 49–50
MAKE: Command that gives a variable any value the user desires, 74–77; practice activities, 77–78
Massachusetts Institute of Technology, 137, 138, 139
Mathland, 139
Memory: Inside the computer where all procedures are stored until erased or until the computer is turned off (also called "workspace"), 18
Mindstorms, 137, 150
MOVE procedure, 98–99
MOVETRUCK procedure, 95–96

Output: That which appears on the screen in response to instructions entered

PACKAGE: Command that saves desired procedures in specific file names without saving every procedure currently in memory, 36–37, 44, 45, 129–130; practice activities, 40, 47, 130
PANE procedure, 21, 63
Papert, Seymour, 137, 138, 139, 143, 144, 146, 148–149, 150, 153
PEAKS procedure, 85
Pen color. *See* SETPC
PENDOWN (PD): Command that causes the turtle to resume drawing (used after PENUP), 11, 12, 22, 115, 119; practice activities, 12, 19, 117, 119, 132
PENERASE (PE): Command that instructs the turtle to erase whatever it is moving over, 12, 116; practice activities, 12, 117
PENREVERSE (PX): Command that instructs the turtle to draw where there are no lines and erase where lines have been drawn, 12, 116; practice activities, 12, 13, 117
PENUP (PU): Command that allows the user to manipulate the turtle without leaving a trail, 11, 12, 22, 115, 119; practice activities, 12, 13, 19, 117, 119, 132
Piaget, Jean, 138
PLAYGROUND procedure, 59
POS: Operation that requests the coordinates *x* and *y* be identified, 72–77; practice activities, 73, 77
POST procedure, 41
PRESENT procedure, 124–126
PRESENTS procedure, 124
Primitives: Refers to commands or basic instructions, 7, inside front cover
PRINT (PR): Command that asks the answer to a forthcoming inquiry be displayed on the screen, 72–76, 101–105; in arithmetic operations, 79, 83, 87
PRINTOUT (PO): Command used for printing out a particular procedure, 22, 32, 39, 131
PRINTOUT PROCEDURES (POPS): Commands that all procedures that have been entered be displayed on the screen, 22, 39, 131
PRINTOUT TITLES (POTS): Commands that all procedure titles be displayed on the screen, 22, 37, 39, 131
Procedures: Used to place a group of instructions into the memory, 17–19, 122; practice activities, 19, 125
Procedures within procedures, 20–22, 124–125
PRODUCT, 79; practice activities, 79–80
PYRAMIDS procedure, 85
PU. *See* PENUP
PX. *See* PENREVERSE

QUOTIENT, 79; practice activities, 79–80

RANDOM: Arithmetic operation that outputs random numbers, 83–86; practice activities, 83
READCHAR (RC): Operation that waits for the user to enter one character or number before continuing a procedure, 104
READLIST (RL): Operation that waits for user to enter an input and press RETURN before executing the next instruction, 105
RECTANGLE procedure, 68–69
Recursive command: Calling a procedure within a procedure by the

same name, 49; practice activities, 50–51
REMAINDER: Arithmetic operation that outputs the remainder of the division problem input, 87; practice activities, 87
REPEAT: Command used when a set of instructions are to be run more than once, 27–33, 60, 95, 114–115; practice activities, 30, 31, 33, 115, 125
RETURN: Key that causes the instructions entered to be executed, 3, 4, 5, 7, 17, 43, 113
RIGHT (RT): Command that turns the turtle to the right the number of degrees indicated, 4, 8, 113; practice activities, 8, 9, 114
Right arrow, 5, 42
ROOM activity, 62–64
ROOM procedure, 21; ROOM1 procedure, 106; ROOM2 procedure, 107; ROOM3 procedure, 107
ROOMGAME procedure, 108–109
ROUND: Arithmetic operation that rounds off the number input to a whole number, 87; practice activities, 87
RT. *See* RIGHT

SAVE: Command used in saving procedures, 35–37, 44–45
Saving procedures, 35–37, 127–129; edited, 44; in packages, 36–37; practice activities, 40; several programs, 128; single programs, 127–128
Screen limits, 51–52; practice activities for setting boundaries, 53, 54
Scrolling: When user requests information that is longer than one screen, the information moves upward and off the screen until the listing is complete, 22
SEASCAPE activity, 64–65
SENTENCE (SE): Operation that puts words into a list and then into a sentence, 104
SETBG: Command (set background) that colors the screen according to the instructions indicated by a code number, 14, 120–122; practice activities, 14
SETH TOWARDS: Operation that instructs the turtle to point in a certain direction by using the coordinates *x* and *y*, 58
SETHEADING (SETH): Command that instructs the turtle to face a specific or absolute heading, 57–58, 72; practice activities, 58
SETPC: Command (set pen color) that instructs the pen to write in the color indicated by a code number, 14, 82, 90, 120–122; practice activities, 14
SETPOS: Command that draws a line from the turtle's location to the specified coordinates, 55–56, 73, 106–107; practice activities, 56
SETX: Command that instructs the turtle to go to the indicated coordinate on the *x* axis, 56–57, 73, 106–107; practice activities, 57
SETY: Command that instructs the turtle to go to the indicated coordinate on the *y* axis, 56–57, 73, 106–107; practice activities, 57
SHOWTURTLE (ST): Command that causes the turtle to appear on the screen, 3, 7, 13, 112, 115; practice activities, 13
SIN: Arithmetic operation that outputs the sine of the input value, 87; practice activities, 88

SPLITSCREEN (CTRL S): Allots the top twenty lines of the screen to graphics and the bottom four lines to text, 16, 17, 109, 118–119
SONG procedure, 102–103
Sprites: Objects that are animated, 95
SQRT: Arithmetic operation that outputs the square root of the input value, 87; practice activities, 88
SQUARE procedure, 28; using SETX and SETY, 56
ST. *See* SHOWTURTLE
STAIRS procedure, 127
STAR procedure, 66–68
STARCOLORS procedure, 82–84
STARS procedure, 70–71, 81
STARTUP file: Series of prewritten Logo files, 34–35, 127, 131
Step: Each tiny movement made by the turtle, 3, 13; practice activities, 6, 7, 114
STOP, 90–94
Stored procedures, 23–26, 51; practice activities, 26, 131–133
Subtract, 79; practice activities, 79–80
SUM, 79; practice activities, 79–80

TABLE procedure, 20, 63
TEXTSCREEN (CTRL T): Allots the entire screen to text, 16, 17, 109, 118–119
Text use, 100–105; lists, 100–101; practice activities, 105; words, 100
TO: Keyword for procedures being committed to the computer's memory, 18, 122
TRIANGLE procedure, 123
TRUCK procedure, 95
Turtle: Small triangular-shaped object in the center of the screen, 3; Dynaturtles, 141; floor, 140; functions of, 141–142; heading and position of, 141; invention of, 139; light, 140; movement, 6
Turtle geometry, 147

UNDERWATER procedure, 96–98

Variable: Refers to a particular number in a procedure, 66–67, 74, 75, 76, 80–84, 87–94, 102; practice activities, 68, 71, 83
Versions of Logo, and vendors, 153

WAIT, 80–81, 106–107. *See also* CTRL W
WINDOW: Command that allows the turtle to go off the screen, but which does not allow it to wrap, 52; practice activities, 53
WORD: Operation that combines elements into one word, eliminating the spaces, 102, 104
Word, 100–101
Workspace. *See* Memory
WORM procedure, 98–99
WRAP: Command that allows the turtle to execute instructions exceeding the screen limits, 51, 52; practice activities, 53
Wrapping around: When commands are given exceeding the screen limits, the turtle keeps coming back on the screen, 29, 51, 113

XCOR: Requests information on the *x* coordinate, 73–76; practice activities, 74

YCOR: Requests information on the *y* coordinate, 73–76; practice activities, 74

Screen Commands

CLEAN	Erases screen without moving turtle, p. 48
CLEARSCREEN (CS)	Erases screen and moves turtle to center, p. 7
CLEARTEXT	Erases all text on screen, p. 17
FULLSCREEN (CTRL L)	Entire screen is allotted to graphics, p. 16
HOME	Turtle is in center of screen, facing straight up, p. 17
SPLITSCREEN (CTRL S)	Top twenty lines of screen display graphics; bottom four lines display text, p. 16
TEXTSCREEN (CTRL T)	Entire screen is allotted to text, p. 16

Editing/Control Characters

CTRL A	Moves cursor to beginning of line, p. 41
CTRL B	Moves cursor back one space without erasing, p. 41
CTRL C	Places procedure into memory, p. 42
CTRL D	Deletes character under cursor, p. 42
CTRL E	Moves cursor to end of line, p. 42
CTRL F	Moves cursor forward one space without erasing, p. 42
CTRL G	Terminates edit mode without making changes, p. 42
CTRL K	Kills (deletes) all from cursor to end of line, p. 42
CTRL N	Moves cursor to line below, p. 42
CTRL O	Opens empty line in procedure, p. 42
CTRL P	Moves cursor to line above, p. 42
CTRL V	Displays next page of procedure, p. 43
CTRL Y	Reinserts most recently killed line, p. 43

Common Error Messages

DOESN'T LIKE ... AS INPUT	Input is not appropriate, p. 82
FILE ALREADY EXISTS	You attempt to save a file under name already used
FILE NOT FOUND	File you are trying to load is not on disk, p. 38
... HAS NO VALUE	Variable named has not been assigned a value, p. 102
I DON'T KNOW HOW TO ...	Instruction is misspelled, pp. 6, 101
... IS A PRIMITIVE	Predefined Logo vocabulary is used incorrectly, p. 62
NOT ENOUGH INPUTS TO ...	More input values are required, p. 66
OUT OF SPACE	Memory is full; erase procedures to continue
STOPPED!	CTRL G is used to stop a procedure, p. 120
TURTLE OUT OF BOUNDS	Fence is in effect and instructions caused turtle to exceed screen limits, p. 52

Arithmetic Operations

ARCTAN (arctangent), p. 86	REMAINDER, p. 87
COS (cosine), p. 86	ROUND, p. 87
INT (integer), p. 87	SIN (sine), p. 87
PRODUCT (*), p. 79	SQRT (square root), p. 87
QUOTIENT (/), p. 79	SUBTRACT (-), p. 79
RANDOM, p. 83	SUM (+), p. 79

Color Codes

0 = Black	3 = Violet
1 = White	4 = Orange
2 = Green	5 = Blue

apple® logo primer

gary g.bitter
nancy ralph watson

Find out what an exciting, useful, and creative language Logo really is! APPLE® LOGO PRIMER makes learning Logo easy for anyone—no previous experience is needed! This handbook covers all the major Logo topics: procedures, recursion, coordinates, the REPEAT capabilities, screen commands, and the Logo library. A step-by-step Logo section for beginners is included as well as a quick-start section for experienced users. Commands, commonly used procedures, and error messages are listed on inside covers for easy reference. With a thorough discussion of all concepts, this book is truly "user-friendly." Suggested activities encourage learning Logo through discovery and exploration. With APPLE LOGO PRIMER, you can discover how easy it is to learn Logo.

The Table of Contents:

STEP-BY-STEP LOGO
- What's Needed to Begin
- Starting with the Turtle
- Developing a Procedure
- REPEAT
- Loading and Saving Programs
- EDITING
- Recursive and Coordinates
- Combining Procedures
- INPUT and Variables
- Arithmetic Operations
- Conditionals, IF, AND, and OR
- Animation
- Text Use
- Combining Text and Graphics

QUICK-START LOGO
AN OVERVIEW OF LOGO

Cover design: Carol Conway

RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company
Reston, Virginia 22090

0-8359-0314-1

Apple® and the Apple logo are registered trademarks of Apple Computers, Inc.